コンピュータ・サイエンス研究書シリーズ ㉚

# オペレーティング・システムの理論

E.G.Coffman, Jr./P.J.Denning 著
名古屋工大教授・工博 川口喜三男
大阪大学助教授・工博 藤井 護 訳

日本コンピュータ協会

コンピュータ・サイエンス研究書シリーズ ㉚

# オペレーティング・システムの理論

E.G.Coffman, Jr./P.J.Denning 著
名古屋工業大学教授・工博 川口喜三男 訳
大阪大学基礎工学部助教授・工博 藤井　護 訳

日本コンピュータ協会

# OPERATING SYSTEMS THEORY

**EDWARD G. COFFMAN, JR.**
*Pennsylvania State University*

**PETER J. DENNING**
*Purdue University*

# ま え が き

## オペレーティング・システムの理論を学習する動機

1969年以来，計算機科学のカリキュラムにおいて，計算機システムの学習は“計算の理論”や“プログラミング”と殆ど同じ位の重要な役割を演ずるようになった．これとは対照的に1965年頃まで，計算機のオペレーティング・システムという研究題目は上記の二つの伝統的な研究分野よりも重要性において劣っていると見なされていたのである．これは重要な情勢の変化である．この変化の最初の兆候は，ACMの**カリキュラム68**†に明示されたことであり，この進展の速さは，全米工学アカデミー教育委員会COSINE(computer science in engineering)会議の作業部会Ⅷの報告である**オペレーティング・システムの原理についての学部コース**(1971年6月)††に早くも明記されている．この変化にはいくつかの重要な理由がある．

第一に，三つの実用上の目的——既存のあるいは将来の設計を改善すること，その正しさと振舞が予め確立されるようなシステムを築くことおよび資源割当て問題と性能評価問題を解くこと——が計算機システムのモデル化についての研究をますます盛んにするように刺激してきた．この努力の主な結果は，オペレーティング・システムの設計と解析の“理論”の出現であった．これは引続いてカリキュラム上に殆ど直接的な衝撃を与えることになった．オペレーティング・システムの諸概念を教える伝統的な“事例研究”の接近法は，未だかつて顕著な成功を収めたということもなく，“モデル化と解析”の接近法に急速に道を譲りつつある．

第二に，複雑なソフトウェア・システムを設計する際の諸問題は，伝統的に，“計算の理論”や“プログラミング”よりも知的な感興に乏しいと考えられていた．いわゆるソフトウェア問題，すなわち，ソフトウェアの開発に要

---

† *Comm. ACM.* March 1968参照.

†† Commission on Education, National Academy of Engineering, 2102 Constitution Avenue, Washington, D.C. 20418.

する高い費用を引下げることおよびソフトウェア・パッケージ全体に対する品質管理を改善することが，ますます厳しく要請されることが分ってきた．その一つの結果として，これまでの伝統は覆されつつあり，深遠な知的問題として，ソフトウェア・システムの設計に関心が強まってきている．“計算機システムの理論”の大部分は，ソフトウェア・システムにおける複雑さを理解し，かつ管理することに何らかの形で関係している．

第三に，我々が生存している世界は，助力を受けずに理解することはとても不可能な程に，複雑な実時間システムになりつつあるという見方がいよいよ広く認められつつある．我々は何度も事業システム，経済システム，社会システムおよび都市システムなどの構築の際に下された決定を見ているが，これらは全て実時間情報システムである．そして，それらの決定は，誠心誠意下されたにもかかわらず，その意図とは似ても似つかない効果をもたらしたことが判明することがよくある．この現象は，オペレーティング・システムの設計者にとっては新しい経験ではない．計算機システムの“理論家”達は，複雑な実時間情報システムを意図した通りに振舞わせるように管理する問題に没頭しており，この問題は，究極的には，計算機産業以外にも衝撃を及ぼすような宿命にあるように思われる．

## 本書の目的

この本の主な目的は，オペレーティング・システムの設計に現われるアルゴリズムを学習することである．この学習には，明確に述べると次のようなものが含まれている．並行処理を可能にしているシステムにおける種々の型の誤動作を避けるために設計された順序付けおよび制御のアルゴリズム，全実行時間および平均流れ時間を最小にするように設計されたスケジューリング・アルゴリズム，多重プログラム化されたタスク間にプロセッサを割当てるアルゴリズム，入出力装置を使うためのアルゴリズムおよび記憶領域を管理するためのアルゴリズムなどである．これらのアルゴリズムは形式主義的立場で学習される．たとえば，与えられた問題について学習するとき，まず，システムに対する数学的モデルを工夫する方法および関心の対象となるアルゴリズムを論ずるであろう．その後に，最適であることの証明，性能尺度の導出，あるいは，システムまたはアルゴリズムがある種の望ましい性質を持っていることの論証などを目的とした解析を行う．

この主題に沿って，本書の教育目標は，オペレーティング・システムのアルゴリズムの研究に適用されてきた最も重要な形式的手法をできる限り単純な形

で適宜に提示することである．我々の興味は，諸結果の性質，解析の本質，それに，その手法の能力と限界などを明確にすることにある．多くの場合，現存するモデルのうち最も単純な形をしたものを選択している．その際に考察をさらに進めて洞察がどれ程深められようとも，そのための一般化が解析の複雑さを倍加するときはいつも，この選択を行っている．オペレーティング・システムの設計を程良く経験した読者が，与えられたオペレーティング・システムを吟味し，かつ，そのシステムについての（形式的な）モデル化や解析を行う研究課題を成功裡に遂行することができるようになる程度までにという基本目的の達成に本書は成功しているのではなかろうか．

本書では，意識的にその取扱いを避けた主題の領域が二つある．それは発見的手法および実験的手法である．（実際，これらの論題と対応する諸結果を含めると本の厚さは確実に2倍になっていたであろう．）スケジューリングにおける組合せ問題に対する発見的あるいはしらみつぶしの探索手法を本書から除外したことは，Conway，Maxwell および Miller の著書† にそれらの問題について優れた取扱いがなされていることによって，大いに正当化される．また，記憶領域管理に関連した実験的な成果を除外したことは，諸文献で包括的に網羅されていることによって正当化されよう．

システムの性能（およびプログラムの振舞）についてのシミュレーションによる研究や統計的手法による解析に関連した実験的な成果は，本書の内容に直結しており，それだけで1冊の本を構成しよう．しかしながら，本書におけるプログラムおよび計算機利用の振舞についてのモデルは，特定のアルゴリズムの性質を調べる必要がある限りにおいてのみ言及している．プログラムの振舞をモデル化した重要かつ良く知られている研究には，並行性に関するグラフのモデルおよび記憶領域の参照に関する局所性のモデルなどがある．本書ではこれらに関して，数学的モデルでなされた諸仮定を支持する種々の技術的研究を頻繁に引用するであろう．

本書は，専門家に対する参考書として，また，学生に対する教科書としていずれにも役立つように体裁を整えている．本書では，読者は計算機科学のカリキュラムの中核において必要とされる諸概念，諸システムおよび諸技法に通じているものと仮定されている．そのため，考案したモデルについての長々しい動機づけや得られた結果についての広範囲にわたる解釈は省略されている．読者には，大雑把にいって学部の4年生あるいは修士の1年生に相当する数学の

---

† R. W. Conway, W. L. Maxwell, and L. W. Miller, "*Theory of Scheduling*" (Reading, Mass.: Addison-Wesley), 1967.

習熟度があることが予期されている．より明細には，(i)応用確率論の諸要目およびマルコフ連鎖の初歩的な知識のほかに，(ii)形式的体系に関する基礎的な学力があることが仮定されている．前者の題材についての簡単な復習が補遺に収められている．

不幸なことに，上述の(i)の仮定のうち，一方あるいは両方は，現在の計算機科学のカリキュラムにおいて極めて頻繁に成立しないように見受けられる．計算機科学の教育の主要な部分がアルゴリズムの研究――アルゴリズムを設計し明確にそれを記述する方法，最適なあるいは効率の良いアルゴリズムの設計の仕方，それに，アルゴリズムの性能を評価する方法――であるということに意見が一致するならば，確率モデルの性質とその応用可能性から，学生は応用確率論および組合せ理論についてのある程度の学力を身につけていなければならないことになる．学生はそのような技法を身につけるまでは，一般的に言って，自明でないアルゴリズムを設計したり，彼が行ったこと――特に，彼のアルゴリズムについて，それがある性質を持っていることを証明したり，その有効性とか性能とかを判定するのに使える尺度を導き出したりしたこと――を正確に他者に伝えたりする能力を持てないということになろう．

各章の末尾に，その章の題材を拡張したものあるいは部分的に変更したものに関連した演習問題を精選して掲載してある．これらの大多数は**演習問題**というよりもむしろ**研究課題**である．読者がこれらを吟味することは特に大切である．なぜならば，これらの問題の目的は，本文に書かれた題材の理解を促進することだけではなく，本書の守備範囲の空白を満たすことにもあるからである．異なる章で取扱った手法を，異なった問題ではあるが密接に関連した応用に拡張するという問題も沢山ある．

## 本書の企画

COSINE 報告によれば，有益な抽象化や理論的展開が引出せるような計算機システムのモデル化には六つの局面がある．すなわち，

手続きとその実現
名前付き対象の管理
保護
並行プロセス
記憶階層の管理
資源割当て

これらの六つの概念のうち最初の三つは，その性質上，より記述的に，すなわち，これらの抽象的概念はどんな数学も必要とせず，実現の際の直接的な手引きとして設計者達に使われるようになる傾向がある．しかしその一方，最後の三つは，有益な結果を得るに先立って数学的な解析が必要である．したがって，本書ではこれらの三つの抽象的概念に注意を限定する．

モデル化と解析が大いに好ましい分野がもっと存在することは確かである．しかしながら，1972年現在では，それらの分野において実用上，重要かつ利用可能な結果はほんの僅かしかないので，本書ではその扱いを全て省いた．それらの中には，システムの信頼性および保全性 (integrity)，システムの性能評価および設計方法論が含まれる．

本書は7章より成っており，各章の概略を次に示す．

**第1章．序論**では，以下の各章における解析の結果が適用できるシステムの物理的性質の概略を述べる．これには，プロセッサおよび記憶装置について本書の内容に関連した諸性質，仮想記憶の実現についての諸問題(特にページング)，記憶管理とプロセッサ・スケジューリングの問題に関する諸局面，それに，システム設計において，並行性，スケジューリング，ページング，資源の共有化およびプログラムの振舞などの概念を使うことの動機付け等々の議論を含んでいる．

**第2章．並行プロセスの制御**では，並行プログラムの制御に関する重要な諸問題，たとえば，決定性，閉鎖，相互排除および同期などを研究するための形式的な取扱いがなされている．これらの問題は，半順序付きのタスク・システムという単一のモデルの環境下で形式化され研究されている．このモデルは，これまでに研究された最も一般的なものでは決してないけれども，現実のシステムで期待される性質を持っている．本書の諸結果に以下のものがある． a) このタスク・システム・モデルは前述の四つの制御問題を統一的に議論できるようにし，それらの問題間の相違と類似性を明確にしている． b) このタスク・システム・モデルは，決定性問題の諸結果の証明を簡単にしている． c) 閉鎖問題の諸結果は諸文献にあるものを拡張したものである． d) 同期問題の諸結果は全て新しいもので，同期用基本操作の一般性と威力を示している．

**第3章．プロセッサ・スケジューリングの決定性モデル**では，この主題について現在利用できる諸結果を完全に近い形で取扱っている．各タスクの実行時間が既知である $n$-タスク・システムおよび $m$ 台のプロセッサが与えられたと

き，最短時間で完了し，あるいは平均流れ時間を最短にし，かつタスク間の先行後続関係の制約を満たしているスケジュールを見出すことが問題である．この種の問題は，古典的なジョブ・ショップ環境下だけでなく，たとえば，第2章で学習した型のタスク・システムが実現されるような将来起り得る環境下でも重要である．

**第4章．計算機における順序付け問題の確率モデル**では，基礎的な待ち行列過程の復習および多重プログラミング・システムにおいてタスクをスケジュールすることへの待ち行列理論の応用が含まれている．できるだけ少ない紙数で，完備した取扱いを示すような試みがなされている．本章で達成される主な目標は，計算機の優先権待ち行列の基本的な解析である．

**第5章．補助記憶および緩衝記憶のモデル**では，特に，入力あるいは出力プロセスに関連して派生する諸問題を取扱う．第4章で開発した手法が広範囲に用いられる．本章には，回転式記憶装置(ディスク，ドラム)の待ち行列のサービスに対する“最短到達時間順”方策の利点(および欠点)の学習もある．また，専用緩衝記憶に相対するものとしての共用緩衝記憶に固有の重大な長所を示す，緩衝記憶利用問題の解析もある．これらの諸結果は，多重プログラミング下のタスク間での任意の資源の共用化，とくに主記憶の分割に関する重要な内容を持っている．この章には，循環型待ち行列網も取扱われている．

**第6章．ページング・システムにおける記憶領域割当て**では，固定された記憶空間における一つのプログラムを管理する際に，与えられた必要時ページング・アルゴリズムに起因するページ往来の制御および解析について知られている諸結果をかなり完全に取扱っている．重要な“スタック・アルゴリズム”の概念についての新しい取扱いと共に，最適アルゴリズムに関する全ての既知の結果も含まれている．また，これらの諸結果と多重プログラミングとの関係が学習される．

**第7章．多重プログラミング下の記憶領域管理**では，“参照の局所性”を示すプログラムの振舞のモデルについての諸性質や多重プログラム化された記憶の管理に関するそれらの関り合いを詳細に取扱う．多重プログラミング下の固定分割および可変分割の方策の諸性質も取扱われる．“作業集合モデル”は，作業集合の大きさ，ページング率，得られるページ参照の型などの間の様々な重要な関係が数節にわたって学習される．

## 謝　辞

本書を準備するに当って，多くの人々から有益な助言を頂いた．特に，辛抱強く激励し続けてくれた Prentice-Hall 社の計算機科学シリーズ編集長 Karl V. Karlstrom 氏と読みづらい原稿を注意深く読んで多数の誤りを訂正し有益な助言をして下さった Richard R. Muntz 氏には，格段の感謝の念を表明する．

そのほか，原稿の部分部分について建設的な批評を頂いた Robert Butler, Robert M. Keller, Stephen R. Kimbleton, W. Frank King Ⅲ, John E. Pomeranz, Barbara Ryan, John Bruno, Vincent Shen, それに Jeffrey R. Spirn の各氏にも謝意を表したい．

本書を作成するに当ってその事務的な面を手早くかつ能率よく殆ど決まりきったやり方でできるようにしてくれた Doris Rice 嬢には特別な感謝を受けて頂きたい．同様に，その助力に対して，Hannah Kresse 女史にも感謝する．

最後に，本書を執筆するに当り，精神的支援および刺激となる環境を与えてくれたペンシルヴァニア州立大学とプリンストン大学に謝意を表する．全米科学財団からは，補助金 GJ-28290 として経済上の支援を受けた．

EDWARD G. COFFMAN, JR.
PETER J. DENNING

# 訳 者 序 文

計算機技術の発達は，とどまるところを知らないかのように見える．1980年代中葉の現在から近い未来を展望すると，オペレーティング・システム(OS)もやはり大きな変貌を遂げるのではないかと思われる．多数のプロセッサを内蔵する並列処理計算機が出現しつつあり，また，極めて多数の計算機と末端装置を相互に接続したネットワークが計画されつつある現下の情況は，それに適合するOSが創出されることを要請しているからである．

ここで，OSの発達の軌跡を振返って見ると，その源がいくつかあることに気が付く．OSにはその機能面の類型がいくつかあるからである．

まず，一括処理システム(batch system)を見ると，この型のOSは，米国ノースアメリカン航空機社によって1958年に開発されたFortran-Ⅱモニタが最初のものであった．その後これを源流として，この種の一括処理専用OSが数多開発され，1960年代中葉までには多重プログラミング制御の最初の経験が獲得された．一方，オンライン実時間システム(on-line real time system)も最初の一括処理システムと同じ時期に開発が進められ，1958年には米軍の作戦指令システム(SAGE)が稼働を開始した．さらにTSS(Time Sharing System)の分野では，米国マサチュセッツ工科大学(MIT)においてCTSSおよびMULTICSが1964年までに開発されている(今日のUNIXはこの流れを継承している)．このような情況からプロセスの概念は1960年代中葉には既に確立していたといえよう．この頃までをOSの発達史における第一段階であると見ることができる．

IBMがOS/360の概念を発表した1964年から1970年代末頃までの時期は第二の段階である．この時期には新世代のOSすなわち一括処理，実時間処理，TSS処理の全てを行える，いわゆる多重モードのOSが開発され，密結合多重処理(shared memory multiprocessing)制御と多重仮想記憶(multi-programmed virtual memory)制御が行われるようになった．

第二の段階のOSに対して，1970年代末頃から疎結合多重処理システムのような形で始まった次の世代が分散処理システム(distributed system)であり，

これは1980年代以降の新世代のOS形態であるといえよう.

以上のようにOSの発達の軌跡を一瞥したとき，本原著の書かれた1972年前後の時期までには既にOSに関する基本的な経験が蓄積され，OS設計の根底にある問題はなにかということはほぼ明確になっていたといえよう.

原著“Operating Systems Theory”は以上のような歴史的背景の下でOSの設計に必要なアルゴリズムを研究する目的をもって書かれた本格的な理論書である．その手法は定量的であり解析的である．内容は，並行プロセスの制御問題，スケジューリング・アルゴリズム，入出力装置や緩衝記憶の効率的利用のためのアルゴリズム，記憶管理アルゴリズム等である．これらは現代のOS設計の立場から見てもますます重要な問題である．訳者の知る限りでは，これだけ広い範囲にわたる主題を扱った類書は本書を措いて他にない．原著の出版から十年余を経た今も，教科書や研究書として十分役に立つと訳者が信じ翻訳した理由がそこにある．もちろん，本書に欠けている重要な理論問題もある．データベース・システムを一分野として含む現代のOSの設計において，情報の保護と秘密保持(security and protection of information)に関する工学的取扱いもその一例であろうし，プロセッサのネットワークと分散処理にまつわる理論問題もある.

訳出に当り，第3章までは主として藤井が，第4章以降は主として川口が担当したけれども，全頁にわたり内容の理解と表現の正確さについてなんども討論し合った．それゆえ担当した部分に関わりなく，両名は連帯して翻訳全体の責任を負うつもりである．また，原著は増刷の際に内容の一部が変更され，その箇所は第2章に集中している.訳注にも記した命題2.3の他，演習問題2.3，同2.13，同2.14などである．これらについては，新しい方を採用した.

訳語については，広く使われている用語を除いて原語の発音に近いとされる片仮名表現を避け，日本語を採用することに努めた．それにもかかわらず片仮名が充満しているのは，訳者の努力不足と言えようか．訳者による造語も少なくない．聞き慣れない訳語のために本書を難解にしているとすれば，その責任は訳者にあるのであって，原著者にあるのではない.

本書の出版に当り，日本コンピュータ協会の西本勝美氏には大変お世話になった.仮名遣いの統一から校正に至るまで多くの作業を快く引き受けて戴いた.ここに記して感謝の意を表する.

1987年3月

川口喜三男・藤井　護

#  目　次

# 序　論　1

## 1.1　オペレーティング・システム

電子計算機の歴史は，1946年から1950年までを第1世代，1950年から1964年までを第2世代，1964年以降を第3世代とする一連の“**世代**(generation)”[1]によって特徴づけられている．この世代という用語は，当初は，ハードウェアの技術の差異を示唆することを意図したものであったが，今では，ハードウェアのみではなく，ソフトウェアも含めた全体としてのシステムに対して適用されるようになってきている[2]．複雑な汎用のソフトウェア・システムの開発は第3世代になって始まり，これが設計や**資源割当て**(resource allocation)に対する理論的接近法の発展のきっかけとなった．

後で詳しく議論するように，“**プロセス**(process)”という用語は，実行中のプログラムを表すのに使われる．計算機システムは，利用者によって生成されたプロセスのために備えている次のようなさまざまな管理と制御の機能によって定義される．

1. プロセスを生成・消滅すること．
2. プロセスの進行を制御すること，すなわち，論理的に進行可能となっている各プロセスが進行し，かつどのプロセスも他のプロセスの進行を無定限に(indefinitely)阻止することはないということを保証すること．
3. プロセスの実行中に生ずる例外条件——たとえば，算術演算誤りまたは機械の誤動作，割込み，番地の不正指定，不法命令または特権命令，保護侵害(protection violation)など——に対応する動作を行うこと．
4. プロセス間にハードウェア資源の配分(割当て)を行うこと．
5. ファイル，エディタ，コンパイラ，アセンブラ，サブルーチン・ライブラリ

やプログラミング・システムなどのソフトウェア資源に到達する(access)方法を用意すること.

6. 情報に対する保護，到達制御(access control)，保全(security)の機能を用意すること.
7. プロセス間の通信と同期の機能を用意すること.

プロセス自身は，これらの機能を必ずしも正確に，あるいは適切に扱うことができるとは限らないので，システムがこれらを提供しなければならないのである．ハードウェアを補佐してこれらの機能を実現している計算機システムのソフトウェアは，**オペレーティング・システム**(operating system)という名で知られている.

計算機システムの形態にも，プログラマや設計者がそれらについて抱く観念にも，非常に幅の広い多様性がある．それにもかかわらず，これらのシステムには多くの共通点がある．これらの共通の特性は，次の三つの一般的な設計目標，すなわち，1) プログラマのために，プログラムの開発，虫取り(debugging)および実行が能率的に行えるような環境条件を整えること，2) 問題解決のための一連の仕掛けを用意すること，3) 資源と情報を共有可能にすることによって計算の費用を安価にすること，に由来するが，その中で最も重要な特性として次のようなものが挙げられる.

1. **並行性**(Concurrency)：同時的に並行して動作する活動(activity)，すなわちプロセスが存在すること，または潜在的に存在すること.
2. **自動的資源割当て機構**(Automatic resource allocation mechanisms)：次のような理由からシステムによって提供される．プログラマは，プログラミング言語を使用することによって機械の細部から遮蔽される傾向があること，活動が並行して行われるシステムでは，資源に対する需要が予測できなくなりがちであること，それに，一つの中枢的な資源割当てプログラムがあれば，システム全体を監視・制御できて，良質のサービスとシステム効率という二つの目的を達成できること.
3. **共有**(Sharing)：二つ以上のプロセスが同じ型の資源を同時に使用すること(その型の資源の個々の単位(unit)が共有可能かどうかにはかかわりなく)，または二つ以上のプロセスが情報を同時に利用すること.
4. **多重化**(Multiplexing)：互いに重複しない間隔に時間を分割し，資源の一つの単位が各時間間隔ごとに高々一つのプロセスに割当てられるようにする技法.

5. システムの資源やプロセスに対して，**遠隔地から会話的に到達すること**(Remote conversational access).
6. **非同期の動作**(Asynchronous operation)：すなわち，事象が発生する順序を予測できないこと.
7. 情報の**長期間にわたる保管**(Long-term storage)：特にファイル・システムの形での保管.

これらの七つの共通の特性の基礎となっている抽象概念は，**プロセス管理と記憶管理**(process and memory management)という二つの広い範疇に分類できる．本書の目的は，この二つの広いクラスの抽象概念に対する理論的な土台を研究することである．我々は，プロセスを管理し(managing)調整し(coordinating)スケジュールする(scheduling)ための，主記憶装置や補助記憶装置を管理するための，それに，計算機システム内の種々の装置間の情報の流れを管理するための**アルゴリズム**の研究に力点を置くであろう．このプロセスの研究には，プロセス間の調停の理論として次の三つの副領域を含んでいる．1) 実行時間が既知の場合の装置割当てに関する最適スケジューリング，2) 実行時間が未知である，あるいは見積らなければならない場合の装置割当てに関するスケジューリング，3) 待ち時間または流れ時間(flow time)が望ましい分布になるようなスケジューリング．これらの副領域は第2，3，4章の主題である．また，記憶管理の研究は，次の三つの副領域を含んでいる．1) 補助記憶装置，緩衝記憶領域および装置間の情報の流れを管理する方策，2) プログラムの命令およびデータのうちどの部分を主記憶に入れておくべきかを決める方策，3) 沢山のプログラムの各々に主記憶を割当てる(多重プログラミングの)方策．これらの副領域は第5，6，7章の主題である．本書の目標は，完全無欠な一般化を目指すのではなく，いくつかの重要な問題と形式的な解析技法を選び出して読者に提供することである.

## 1.2 資　源

本書で得られる結果は，広範囲にわたる計算機システムに適用される．これらのシステムは，以下に述べるようなハードウェア資源(装置)の性質についての一般的な仮定を満たしているとする.

情報の変換作用を行う，すなわちプロセスの進行を遂行する任意の装置を**処理装置**(または**プロセッサ**，processor)という．命令を実行してデータに算術演算を行う演算処理装置(arithmetic processor)，または中央処理装置(central

processing unit)の通常の概念は，この定義の範疇に入る．それほど良くは知られていないが，入出力プロセッサ(input/output processor)，またはチャネル(channel)の概念もこの定義の範疇に入る．

情報の保管のために使われる任意の装置を**記憶**(memory)という．半導体記憶，磁心記憶，ディスク，磁気ドラム，磁気テープなどがその例である．記憶装置の**容量**(capacity)は，それが蓄えることのできる情報の語(またはバイト)数である．記憶装置の**到達時間**(access time)は，要求の受取りから完了までの平均の時間間隔であり，待ち行列における遅延は含まない．与えられた記憶装置内で語を保管する場所の名前を**番地**(address)という．指定された番地の保管場所と読取り/書込みヘッドとが接近するように読取り/書込みヘッドをその保管場所まで移動させるか，または保管媒体を動かすような機械的な動作が一般には必要であるとき，そのような記憶装置は**空間的に番地付けられる**(positionally addressed)という．そのような記憶では，語への到達時間は記憶媒体中のその語の位置に依存する．各記憶場所への到達時間が同じであるとき，その記憶装置は**等時間到達**(random access)であるという．ディスク，磁気ドラム，磁気テープは前者の例であり，半導体記憶や磁心記憶は後者の例である．

計算機システムでは，いつも，**主記憶**(main memory)と**補助記憶**(auxiliary memory)という少なくとも2レベルに構成された，様々の異なった型の記憶媒体を用いてきた．情報が処理のために直ちに取出されるようにするためには，主記憶——典型的には，到達時間が非常に短い等時間到達のもの——になければならないし，そうでないものは補助記憶——典型的には，到達時間が比較的長い，空間的に到達するもの——にあってもよい．大抵の等時間到達の記憶は，語の保管場所を番地指定するために何らかの形の電子的な切換え機構を用いており，このために到達時間は非常に短くなっている．半導体記憶の典型的な到達時間は100 nsecであり，磁心記憶の場合は1 μsec(記憶サイクル時間)である．また，大抵の空間的に到達する記憶の典型的な到達時間は，一番速い磁気ドラムの10 msecから磁気ディスクの200 msec以上という値の範囲にある．記憶の大容量化を求める欲求は，主記憶と補助記憶について，その到達時間と容量との間で，これまでいつも妥協することを強いられてきたのである．

第5，6，7章で重要であるから，ここで主記憶装置と補助記憶装置の到達に関する諸性質を復習しておこう．記憶の割当てと転送の単位としては，個々の語よりも**ブロック**(block)を使うのが普通である．ブロックは一意に定まって

いる名前を持ち，番地が連続しているいくつかの語から成っている．全てのブロックが同じ大きさであるとき，ブロックのことを**ページ**(page)と呼ぶ．主記憶の語に対する参照要求は，絶対番地の形か，対$(b, r)$の形で(プロセッサによって)提示される．ここで，$b$ はブロック番号，$r$ はブロック $b$ の基底番地からの相対的な番地である．補助記憶への要求の仕方には，(ブロック内の語に到達するというよりもむしろ)ブロック全体の転送を求めるのが典型的である．このブロックの転送要求は，$p$ をそのブロックの先頭番地，$l$ をそのブロックの長さとしたとき，対$(p, l)$の形をとる．

**磁気ドラム記憶**(drum memory)の構造を図1.2-1に示す．ドラム記憶は，一定の速さで回転している円筒で，$T$ 時間単位ごとに1回転する．このTの典型的な値は30 msecである．語の保管場所は，$t$ を**トラック番号**(track index) $\theta$ を角位置としたとき，ドラム上の座標点$(t, \theta)$である．各保管場所は，1回転ごとに1回，読取り/書込みヘッドの下を通過するから各語は $T$ 時間単位ごとにただ1回だけ到達できる．**トラック**(track)は，図1.2-1に示すように，ドラム上，1周分の帯状の保管領域である．(通常，伝送路を他のトラックに切換えるのに若干の時間が必要であるが，この時間は普通の場合，本書の解析では無視してもよい程，十分小さなものである．)このように，ドラム上のブロックに対する要求は，ブロック長を $l$ とすると($l$ は1トラックの語数を越えない)，$(t, \theta, l)$の形で行われる．全てのブロックが同じ長さで，ブロック長

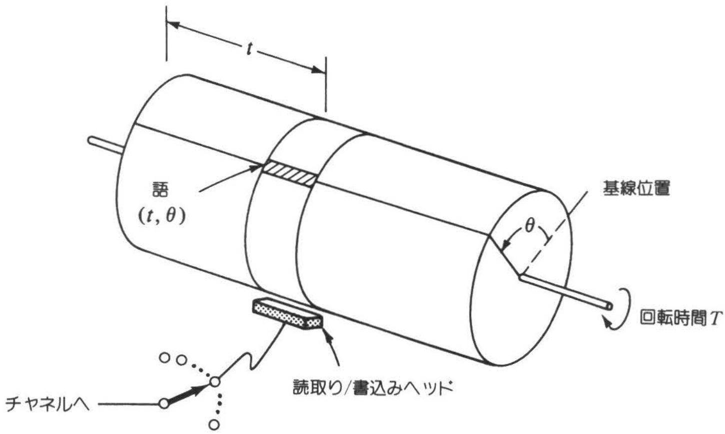



**図1.2-1**　ドラム

$l$が予め分っているページングの場合には(図1.2-2)，角位置は**セクタ番号**(sector index)として与えられ，要求はより簡潔な$(t, \theta)$の形をとる．図1.2-3は，ドラムに対する要求が蒙る遅延を示している．行列内待ち時間は要求が待ち行列内で待っている時間を，回転待ち時間はドラムが望ましい角位置まで回転する時間を，転送時間は要求されたブロックの語を転送するための時間をそれぞれ表している．

**磁気ディスク記憶**(disk memory)の構造を図1.2-4に示す．何枚かの円形の板(円盤，disk)が共通の軸に装着されており，その軸は$T$時間単位ごとに1回転という一定の速さで回転する．この$T$の典型的な値は60 msecである．動作方向が一次元的で，その方向に沿って存在する$k$個の位置の範囲内を動くような可動アーム(ヘッド支持腕，arm)がある．$t$をトラック番号，$d$を**ディスク面番号**(disk surface index)，$\omega$を角位置としたとき，記憶の各保管場所は，座標$(t, d, \omega)$を持つ．**トラック**(track)は，与えられたディスク面の与えられたアーム位置の語の集合であり，**シリンダ**(cylinder)は，与えられたアーム位置の全ディスク面の語の集合である．アームが与えられたシリンダに固定されたときのディスクの性質は，ドラムの性質と同じであるということは注目に値する．実際，ある製造会社のドラムは，可動のアームがない代りにシリンダ毎に恒久的に置かれたアームを持つディスクとして実現されている．このよ

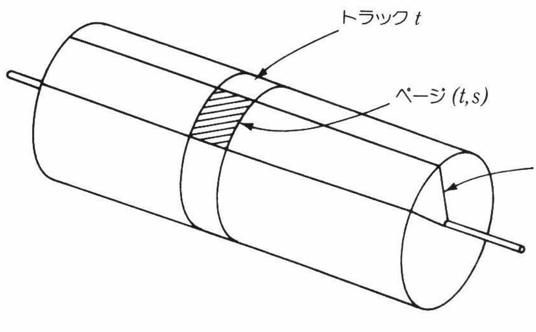



図1.2-2　ページドラム

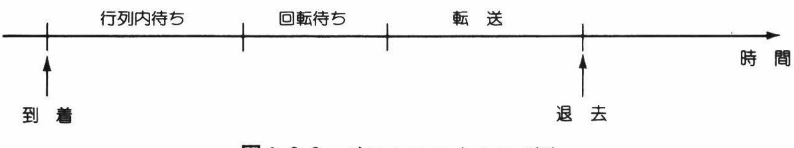



図1.2-3　ドラムのタイミング図

うな装置は，一連の同心円のドラムを定義しているものと見なすことができる．図1.2-5に，要求に対するタイミングの図を示す．ここで，**探索時間**(seek time)は，要求されたシリンダまでアームを移動させるのに必要な時間である．与えられたディスク装置の探索時間の典型的な範囲は，最短(非零)の移動に対する120 msecから最長のそれに対する250 msecであるから，探索時間は回転

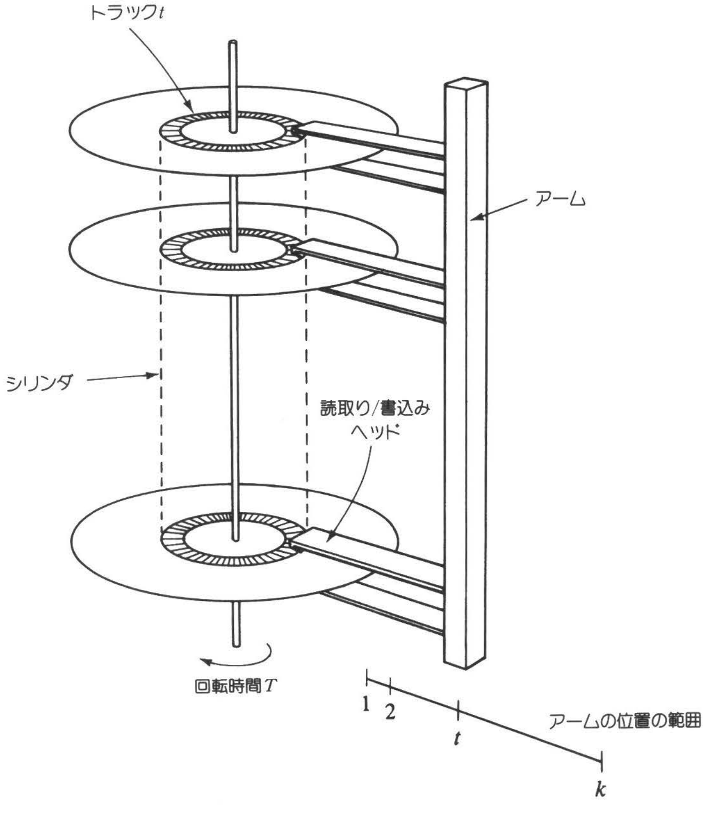



図1.2-4　ディスク




図1.2-5　ディスクのタイミング図

待ち時間よりも支配的である．

これまでの議論では，本書の読者に興味のあるハードウェア装置の重要な特性を扱ってきた．第2章の資源割当て問題の議論では，上述のようなハードウェア資源よりももっと一般的な意味で，“資源”という術語を用いる．資源という非常に一般的な見方を採用することにより，オペレーティング・システムの多くの問題を，意味のある結果が得られる資源割当て問題として扱うことができるようになる．この見方に従うと，プロセスが使用することができ，かつ，プロセスの進行がそれに依存するような対象の集合は，そのプロセスによって要求される**資源**(resource)の集合と呼ばれる．システムはさまざまの資源の**型**(type)を備えており，一般に各型ごとに，限られた数の利用可能な**資源単位**(resource unit)がある．資源の型の例として，プロセッサ，記憶，周辺装置，データ・ファイル，手続き，メッセージなどがあり，資源単位の例として，プロセッサ1台，記憶1ページ，ディスクまたはドラムの1トラック，ファイル1個，手続き1本，メッセージ1通などが挙げられる．資源の型には，ハードウェアによって直接実現されているもの(たとえば，プロセッサ，記憶，周辺機器)もあれば，ソフトウェアによってのみ実現されるもの(たとえば，ファイル，手続き，メッセージ)もある．また，資源の型の中には，資源単位が解放(release)された後で再び割当てることのできる**再使用可能**(reusable)なものもあるし，これとは対照的に，使用後は消滅する**消費的**(consumable)なものもある(たとえば，メッセージ，呼出された手続きの活性化レコード(activation record))[3]．全ての資源は，その利用者への情報を担っている内部的な“状態(state)”を持っている．もし資源の状態が使用されている間に変更されるならば，この型の単位は，一時には高々一つのプロセスによる“排他制御(exclusive control)の下におかれるように制約されている”といわれる．

プロセスから資源を横取り(preemption)するときは，その資源の状態を保存しておく必要がある．後で状態を横取りされた時の状態に復元することによってこの資源を元のプロセスに返すことができる．資源の各単位には，時刻 $t$ においてそれを使用中のプロセスから横取りする際に要る費用を表す $t$ の関数(**横取り費用関数**(preemption cost function)という)が附随していると考えることができる．時刻 $t$ におけるその費用がある閾値を越えていれば，その時点ではその単位は横取り不可，そうでなければ横取り可と考えられる．横取りの結果，入力や出力や進行に損失が生ずるならば，より一般的に言うと，資源の状態を保存するのに多大の努力が必要であれば，横取りの費用は高いと考えられる．よくあることであるが，資源単位が時刻 $t_i$ には横取りされ得るけれど

もその時以外には横取りされないというような時刻の系列 $t_i$ が存在することがある．たとえば，プロセッサに対しては各命令実行の終了時が，ドラムに対しては要求の退去時が，また，ラインプリンタに対しては印刷の完了時がそれぞれ $t_i$ に対応する．$t_i$ は，前もって知られている場合(たとえばプロセッサに対する場合)もあれば，資源単位が横取り可能になったことをプロセス自身が知らせなければならない場合(たとえば，プロセスが資源を解放する場合)もある．

## 1.3　並行プロセス

第3世代の計算機システムの開発は，プログラムの設計法に関して，**並行(並列)プログラミング**(concurrent (parallel) programming)という新しい局面をもたらした．この理由として次のものが挙げられる．

1. 速い応答時間と資源の効率的な利用を求める欲求によって，種々の形態の多重プログラミングが導入され，動的な各プログラムの“状態”と資源に対する要求を明示する必要性が生じてきた．
2. 装置類を有効に利用したいという願望から，中央処理装置および諸周辺装置間の並行活動をますます多用するようになった．
3. 実行中のプログラムの間で，情報を共有したり通信したりしたいという願望から，同期についての明細の指定をソフトウェア・レベルで行う必要が生じてきた．
4. オペレーティング・システムはさまざまな並行活動を調整する制御プログラムであると見なすと便利であることが分ってきた．

たとえハードウェアの技術が並行活動の実現を許したとしても，実際にそうなるようにプログラムを書くという仕事がオペレーティング・システムの設計者に残されている．この並行活動の重要なモデルを研究することが第2，3，4章の目的である．

ある着目した瞬間にそのプログラムの命令をプロセッサが実行中であるか否かにかかわりなく，“実行中のプログラム”という概念が任意の時点で意味を持ち得るように，プロセッサの活動の抽象概念としての術語である**プロセス**(process)は1960年代前半のいくつかの開発計画(たとえば，Multics や TSS/360)で現れた．これらのシステムでは，プログラムが命令の系列であるのに対し，プロセスはプログラムによって遂行される作用(action)の系列であることに着目して，プロセスとプログラムは別物とされた．プログラムは静的でありプロ

セスは動的である．この区別のそもそもの目的は，計算の結果に影響を与えずにプロセッサを横取りしたりプログラムを再開したりできるようにすることであった．**並列プロセス**(parallel process)の概念は，与えられた任意の時点で二つ以上のプログラムが開始後でかつ終了前の状態にあると認められるということを意味すると解釈される．このとき，プロセスは同時に進捗しつつあってもよく，そうでなくてもよい．たとえば，与えられた時点において，プロセスは中央処理装置上で走行中であり，別のプロセスは中央処理装置割当ての順番を待っていて保留中であり，第3のプロセスは入出力チャネル上で走行中であるという状態であってもよい．プロセスは二通りの方法で相互に影響を及ぼす．一つは，同じ資源を競合することによるもので間接的であり，他の一つは情報を共有したり通信したりすることによるもので直接的である．プロセス同志が論理的に無関係であるとき，間接的に影響を及ぼし合い，並行制御はそれらのプロセスが乗っているシステムに委託される．しかし，プロセス同志が直接的に影響し合っている場合，その間の並行制御はそれらのプロセスを実現する(すなわち，プログラミングの)段階で陽に表現されていなければならない．

不幸にもこの分野の文献では，プロセスについて，互いに矛盾するものすらあるほど数多くの異なった定義がある．たとえば，Dennis と Van Horn [4] が提案した“命令の系列内での制御の軌跡”というのがある．Saltzer [5] の“疑似プロセッサ上で実行中のプログラム”，Van Horn [6] の“手続きのステップを実行している事務員”，Horning と Randell [7] の“プログラムの状態の系列”などもある．また，先刻提案した“実行中のプログラム”や，“プログラムは命令の系列であり，プロセスはプログラムによって遂行される作用の系列”というのもある．IBM のシステム/360を設置したいくつかのシステムで使われているように，“プロセスはプログラムの状態表(program status block)である”というのもある．**プロセス**を定義しようとするあらゆるこれらの試みの結果の一つは，プロセスという術語は，システム内のどんなものであれ，任意の計算活動を表すような極めて一般的な意味を帯びるようになってきたことである．もう一つの結果は，上述の全ての概念と矛盾しない単一の定義を採用することは殆ど不可能であるということである．そのため本書では，全ての形式的な議論で用いるために，より当り障りの少ない術語である**タスク**(task)を採用する．一般論，あるいは形式的でない議論では引続き“プロセス”を用いることにする．タスクは，計算の不可分な単位として用いられ，その外部との境界における振舞は明示されるけれども，その内部における操作は明示されず議論する興味もない．応用によっては，タスクは命令にも，手続きにも，システ

ム全体にさえもなり得る．

$P(\mathbf{x}, t)$は，時刻 $t$ におけるシステム変数 $\mathbf{x}$ の値に関する言明(assertion)を表す述語であると仮定する．$P$ が成立するとき，システムによってある特別な動作が引起されると仮定するならば，“$P$ は真である”ということがシステムにおけるある**例外条件**(exceptional condition)の成立を定義している [4]．$P$ が真である期間の始まりは，すなわち条件 $P$ の発生である．例外条件の例として，不法命令の実行を試みること，算術演算の溢れ(overflow)，番地の不正指定や周辺装置の処理の完了などがある．例外条件には，(上述のように)ハードウェアで検出されるものもあるが，ソフトウェアによっても検出され得る(プロセス間メッセージや PL/I のプログラム割込みのように)．例外条件を表すのに，これまでしばしば意味に矛盾が見られる多くの術語が使われてきた．例外(exception)，割込み，動作異常，割出しなどである．**動作異常**(または**動作例外**，fault)という術語は，与えられたプロセッサ上を走行中のタスクの動作の結果として生じた例外条件に対して最も多く使われている．**割込み**(interrupt)は，与えられたプロセッサ上を走行中のタスクとは別のタスクによって引起される例外条件に対して最も多く使われる．**割出し**(trap)は，例外条件の発生に対応してとられる動作を表すのに最も多く使われる．システムが認識する各々の例外条件に対し，その条件の**ハンドラ**(handler)と呼ばれるタスクがあり，その条件の発生後，短時間のうちに呼出されるものと仮定されている．各ハンドラは，“起こされる”べきプロセスとして，すなわち，呼ばれるべきサブルーチンとして実現することができる．殆どのシステムは，走行中のプロセスからプロセッサを横取りし，そのプロセッサを適切な割込みハンドラのタスクに割当てるために，特別な割込みハードウェアとソフトウェアの組合わせを持っている．ハンドラ・タスクが正しく動作することを保証するために，そのようなタスクの走行中，プロセッサは**割込み不可モード**(uninterruptible mode)(“割込みは遮蔽される”)になる．このように，ハンドラ・タスクの動作は，システム内の他の動作との関連において“不可分(indivisible)”である．あるシステムでは，例外条件には(その条件に素早く応答することの相対的な重要さに基づいて)優先順位が割当てられており，現在動作中のハンドラの優先順位よりも高い優先順位の条件が発生すると，高順位の条件のために，低順位のハンドラに対する割込みが引起こされる．ハンドラ・タスクが完了すると，システム・スケジューラ(system scheduler)は，何か他のタスク——優先順位が与えられているシステムでは，もちろん，保留中のハンドラ・タスク——を再起動し，プロセッサは全てのハンドラ・タスクが完了した後でのみ，割込み可

能モードになる．

### 1.3.1 プロセスの調停

第2章には，プロセスの制御に関する互いに関連した四つの問題が含まれている．これらの問題は，タスク間に処理の先行後続関係に関する制約(precedence constraint，以下，先行制約という)が規定されているタスク・システムを用いて記述される．

1. **決定性**(Determinacy)．共通のデータを用いるタスク・システムによって遂行された計算結果が各タスクの相対的な速度に依存してはならないこと．
2. **閉鎖**(Deadlock)．次のような状況は避けるべきである．並列タスクのある集合に属する各タスクが互いに自分の保持している資源を確保したまま，他のタスクが保持している資源を要求してそれが解放されるのを待つという"循環待ち(circular wait)"に陥るような状況．
3. **相互排除**(Mutual exclusion)．あるいくつかのタスクは，それらが同時に実行されることのないようにプログラムされていなければならない．たとえば，あるタスクがファイルを読んでいる最中に他のタスクがそのファイルに書込んではいけない．
4. **同期**(Synchronization)．あるいくつかのタスクは，他のタスクの制御下にある何らかの事象(event)が発生するまで，与えられた点で停止していなければならない．

文献において大いに注目を集めた最初の問題の一つは，決定性問題であった．二つ以上のプロセスが共同して一つの問題の解を得ようとするときにはいつも，計算の結果がそれらのプロセスの相対的な速度に依存する可能性がある．多重プログラミング・システムにおいては，プロセスへの割込みが任意の時点で発生することや，周辺装置が任意の速度で並行的に動作することを許容しなければならないから，計算が速度に依存しないように，すなわち**決定性**になるように，協同作業を行うプロセスを規制することが必要である．速度に依存しない論理回路の研究は1955年から1960年の間にMuller [8] によって行われていたにもかかわらず，プログラミングにそれを応用することはほぼ10年後[9, 10] まで評価されなかったということは興味深い．

もっと最近になって，プロセス間の干渉に関する様々の問題(閉鎖，相互排除，同期，メッセージ送信を含む)が大いに注目を集めるようになった．それ

らに対する種々の解は資源割当て手続きの改善と，とりわけ入出力制御と割込みシステムの改善を先導することになった．相互排除問題の解は Dijkstra[12] と Knuth [10] によって初めて与えられた．並列性(parallelism)の抽象的モデルにおける同期問題の扱い方は Karp と Miller [105] に固有のものであった．後に Dijkstra [11] は，具体的なプログラミングの用語法で同期問題を扱い，オペレーティング・システムへの応用に対して良く知られた PV 操作(P and V operations)を考案した．閉鎖問題の考察は，Dijkstra [11]，Habermann [13]，Havender[14]，Shoshani[106] や Murphy[15] から始まった．並列性に関するかなり広範な数学的知識が 1960 年代の初期に集積されたということは注目すべきである．たとえば，Petri [16] の非同期網(asynchronous nets of Petri)があり，Karp と Miller [17]，Luconi [18] や Slutz [19] の並列プログラム形(parallel program schemata)があり，Rodriguez [20] のプログラム-グラフの理論 (program-graph theory)や Holt と Commoner [21] のマークグラフの理論(marked-graph theory)がある．これらの研究は全て，本書では扱えない程の深みに達している．

### 1.3.2 タスク・システムのスケジューリング

第 3 章には，タスクを実行するのに利用できる $m$ 台の(同一の)プロセッサが与えられたとき，各実行時間 $\tau_i$ が既知であるタスクの半順序集合(第 2 章のタスク・システムのモデルを用いる)のスケジューリング理論の成果が記されている．スケジュールの性能についての二つの尺度が考えられる．最後のタスクが完了するまでの時間および平均待ち(流れ)時間(average turnaround (flow) time)である．第 1 の尺度は，プロセッサの利用度との関連から重要である．実際，与えられたスケジュールが時間 $T$ 内に終了するならば，そのスケジュールによるプロセッサの利用率(utilization factor)は，次式で与えられる．

$$u = \frac{1}{mT}\sum_i \tau_i$$

明らかに，$T$ を最小にすれば $u$ は最大になる．第 2 の尺度は，利用者が経験するシステムの性能に関連するので，また，(その尺度を最小にすると)未完了タスク数の平均値がしばしば最小になるので重要である．読者は，両方の尺度で見て同時に旨く行くスケジュールを見つけることは，多くの場合困難であるということが分るであろう．さらに，最適スケジュールを生成するための簡単

な(数えあげ法ではない(nonenumerative))手続きは，タスク・システムに極めて強い制約を課した場合——たとえば，全ての(タスクの)実行時間が等しいとか，タスク・システムが木構造になっているとか——にのみ存在し，かつ，このような特別な場合でも，その手続きが本当に最適スケジュールを生成することを証明するのは骨の折れることであるということにも読者は気付くであろう．

第3章に示された類の結果は，現今のシステムへの適用性としては限られていることは明らかである．特に，タスクの実行時間$\{\tau_i\}$を実行前に知ることは普通は不可能であり，精々$\tau_i$の確率分布を知ることができる位である．もちろん，これに対する反論として，実際のスケジューリングの手続きが最適のそれと比較してどの程度旨くできているかを，これらの結果を用いて決定することができるということはできる．第3章の結果には，二つの型のスケジュール長の比について述べたものが多い．たとえば，任意のタスク・システムを持つ$m$台のプロセッサについて，任意のスケジュールの長さと最適スケジュールの長さの比は，$(2m-1)/m$を越えることはないというのがある．この例は，現実のスケジューラは，最適スケジュールを処理する効率に比べ，$m/(2m-1)$倍よりも悪くなることはないということをシステムの設計者に教えており，そのスケジューラを改善する場合の手引きを与える．

第3章で考察するスケジューリング問題のクラスは，計算機科学との関連で考察されるよりも何年も前からある全く古い問題である．これらの問題は，最初はオペレーションズ・リサーチの研究者にとって長年の間興味の対象であった古典的なジョブ・ショップ(job-shop)問題や組立てライン(assembly-line)問題として考察された．最も古い重要な研究の一つは，1954年のJohnson [22]による2-および3-段の(two- and three- stage)組立てライン問題についての今や古典となった研究である．ジョブ・ショップに関連するタスクの半順序集合について，経営科学やオペレーションズ・リサーチの分野ではかなりの量の研究があり，“臨界径路(critical path，最長の順序付けられたタスクの実行時間の系列)”の解析や，先行後続関係(以下，単に先行関係)の構造(precedence structure)とか，資源の使用可能性(可用性，availability)または費用とか，タスクの実行時間とかの変化に対する感度(sensitivity)などを取扱っている．代表的な教科書として，Clark [23]による1952年のガント図(Gantt chart)についての本や，PERTの解析に関するModerとPhillips[24]の1964年の教科書がある．

最適スケジュールに関する簡明な結果を求める努力はいくつかある．まず，

上で引用した Johnson の研究がある．1959 年に McNaughton は，横取り権(preemption)のある場合で，納期(deadline)に間に合うように，独立のタスクをスケジュールする問題についての結果を発表した [25]. 1961 年には Hu が，タスクの実行時間が等しい場合について，先行関係が木構造であるタスクの集合を最適にスケジュールする今や良く知られている結果を発表した [26]. 1967 年に，Manacher はスケジュールにおける変則的な振舞を制御することについての技巧的な研究を発表している [27]．すなわち，たとえばタスクの実行時間が減少したり，プロセッサの台数が増加したとき，最適スケジュールの長さが**増加**することがあり得るのである．1969 年, Graham はこのような変則的事態の効果の限界を発表した [28]．1969 年に，Fujii 等は，プロセッサが 2 台で，各タスクの実行時間が等しい場合について，タスクの任意の半順序集合に対する最適スケジュールを見出す方法を発表した [29]．そのアルゴリズムは，2 部グラフ(bipartite graph)† の最大マッチングを用いており，$n$–タスク・システムに対してその最適スケジュールを求めるのに少なくとも $n^3$ に比例する時間が必要である††．1972 年，Coffman と Graham は同じ問題に対して $n^2$ のアルゴリズムを発表した [30]．1970 年には，Muntz と Coffman が任意のタスク・システムに対し，横取り権がある場合の最適スケジュールを求める方法を発表した [31]．横取り権ありのスケジュールに対しては, 多重プログラミング環境への応用から，近年，興味が持たれるようになってきた.

これらの最適スケジュール問題に対する成果が少ないことから，最適スケジュールを作り出す数え上げ法ではない速いアルゴリズムを見つけることは困難であることが推論される．Conway, Maxwell と Miller は，1967 年の優れた教科書 [32] の中でこのことを詳しく論じており，この問題についての相当量の参考文献を記載している．1971 年，Krone は最適に近い(near-optimal)スケジュールを効率良く作り出す発見的(heuristic)手法を導いた [33].

### 1.3.3 スケジューラの確率モデル

第 4 章では引続きスケジューリングの手続きの研究を行う．第 3 章の研究の基本条件は，タスクの実行時間が前もって分っており，全てのタスクは直ちに実行可能になるという前提であった．第 4 章では，これらについての確率的記

† **訳注** 正しくは，2 部グラフではなく，一般の無向グラフにおける最大マッチングを用いている.

†† **訳注** 正確には，“$f(n)$を最大マッチングを求めるアルゴリズムの手数($\mathcal{O}(f(n)) \leq n^4$ は既知)としたとき，最適スケジュールを求めるのに必要な手数は，高々 $\max(n^3, f(n))$ のオーダーである”.

述のみがなされる場合のスケジュール技法の性質を研究する．

1960年代の初期から中期にかけて，非常に多くのシステム設計者達は，**優先権スケジューリング**(priority scheduling)の考えを採用した．各時点で各タスクはそれぞれある優先度を持っており，横取りされるか，またはタスクがプロセッサを解放することによりプロセッサが遊休状態になるときはいつも，スケジューリング手続きは，優先度が最も高い動作可能なタスクにプロセッサを割当てるのである．いくつかのシステムでは，どんなタスクも不当に長くプロセッサを保持することがないように“時間切片(time slicing)”を用いている．この場合スケジューラは，“インタバル・タイマ(interval timer)”に時間量子(time quantum)を設定し，タスクが時間量子を使い切ったのに自発的にプロセッサを解放しないならば，タイマは割込みを引起こしてそのタスクからプロセッサを横取りし，新しいタスクにそれを割当てられるようにする．このスケジューリング原理は，1968年にLampsonによって概説されている[34]．タスクの優先度は，タスクがシステム内にある間は変更されないかも知れないし，タスクの観察された振舞(それまでの履歴)に基づいて時々刻々計算し直されるかも知れない[35]．

第4章の論じ方は，待ち行列理論(queueing theory)における基礎的な概念と手法の復習から始まる．指数分布とその性質，FIFO(先着順, first-in-first-out)待ち行列システム，待ち行列の長さと待ち時間とを関係付けるリトルの公式，それに，入力と出力の分布のパラメータと待ち行列の長さを関係付けるポラチェックーヒンチン(Pollaczek-Khintchine)の公式などである．章の残りでは次のようないくつかの重要な待ち行列のクラスの解析を紹介する．

1. **ラウンドロビン**(round-robin)．これは，初期のタイムシェアリング・システム(時分割システム，time-sharing system)で使われた待ち行列である．タスクはFIFO待ち行列に並ぶ．その行列の先頭に来る度に，タスクは1時間量子の限度内で，可能な限り長くプロセッサ上で実行される．完了しなかったならば，そのタスクは(プロセッサを)横取りされ，待ち行列の最後尾に置かれる．
2. **単純優先権待ち行列**(simple priority queue)．各タスクは到着時に割当てられる一定の優先度を持っている．プロセッサが空くと必ず優先度が最高のタスクを走らせる．横取り割込み可のシステムもある．すなわち，走行中のタスクのものよりも高い優先度のタスクが到着すると，前者は(プロセッサを)横取りされ後者にとって代られる．

3. **最短経過時間順**(shortest-elapsed time)**および最短処理時間順**(shortest-processing time)**待ち行列**. これらの待ち行列は，実行時間の短いタスクを優遇することを試みるもので，到着以来，最もプロセッサを使っていないタスク，または全実行時間が最短のタスクに優先権を与えることによってこれを行う．新しいタスクが到着すると，走行中のタスクからの(プロセッサの)横取りが生じることもあり得る†.

第3のクラスの解析は，これらの待ち行列が横取り割込み可の優先権待ち行列としてモデル化できるし，他の二つのクラスの手法も使えるので，簡単である．このクラスは，1962年にCorbato等によって報告され，良く知られている，CTSSのスケジューラのための多重レベル待ち行列(multilevel queue)を含んでいる [36]．全ての結果は，時間量子の大きさ $Q$ で表される．$Q \to \infty$ の場合は横取り割込み不可の場合に対応し，$Q \to 0$ の場合はいわゆるプロセッサ共有(processor-sharing)の場合に対応する．

待ち行列の理論は，Erlangが一連の論文を発表した1909～1918年の間に始まった．これらの論文では，状態が通話中の呼の数を表し，呼の到着間隔およびサービス時間が指数分布に従っているようなマルコフ過程(Markov process)によって電話系統がモデル化できるということが証明されていた．1930年代初期のPollaczekとKhintchineの研究は，**確率的サービス・システム**(stochastic service system)についての知識の発展に大いに寄与した．1950年代は，オペレーションズ・リサーチの研究者達によって，確率的サービス・システムへの関心が文字通り爆発的に広められた．これらのモデルは，自動車交通の輻輳(ふくそう)とか，生産と流通における待ちとか，商業における在庫管理のような問題を解析するのに有効であることが分ってきた．1960年代の初期は，これらの結果を計算機のスケジューリングの問題に適用しようという最初の真面目な試みが始まった時代である．これらの試みは，その提案者達が望んだようには必ずしも成功しなかったが，これは主に次の理由による．計算機のスケジューリングの方法は，有限の入力母集団(finite source population)で，それまでの型にはまらない待ち行列の構造を持ち，複数個の輻輳点や異なった型のプロセッサがあるシステムであり，横取り割込み可のスケジューリングである――このどれもが，それまではあまり注目を受けていなかったのである．

計算機技術者にとって重大な関心のある最初の解析は，無限の入力母集団に対するラウンドロビン待ち行列についての，1964年のKleinrockの論述であ

† **訳注** 特に最短処理時間順の場合に対しての話である．

る[37]．同じく1964年には，有限の入力母集団について，より現実的な仮定に基づいて同じ問題を論じたCoffmanとKrishnamoorthiの論文もある[38]．1966年にKrishnamoorthiとWoodは，到着間隔とサービス時間が指数分布している場合のタイムシェアリング・システムにおけるスケジューリングについて，すっきりした論述を示した[39]．先程注意したようにCorbato等は1962年に，CTSSに対する多重レベルの帰還型待ち行列(feedback queue)を提案したが，その目的はシステム内でのジョブの入換え(swapping)の経費(オーバヘッド)† を制御することであった．この待ち行列システムは1965年にSchrage [40]によって解析され，CoffmanとKleinrockがその諸変形をさらに研究した[41]．1965年にScherrは，待ち行列の理論における古典的な機械の修理モデル(machine-repair model)を用いることによって，CTSSの性能を十分な信頼度で予測できることを示した[42]．

待ち行列システムにおける優先権の研究は新しいものではないけれども(1954年のCobham[43]参照)，計算機への応用に対する関心は，優先権スケジューリングの方法を拡張して使うことによって生じてきた．システム設計者がプロセッサを(ラウンドロビンのように)多重化利用する技法についての経験と自信を得るにつれ，極端に時間量子が短い場合のシステムの極限(これは，“プロセッサの共有”という極限として知られているが)の振舞にかなり興味が湧いてきた．この興味は主に，パラメータ $Q$ の値を零に近づけることによって達成される解析上の簡潔さから生じたものである．上に引用した解析の多くは，時間量子の大きさを零に近づけたときの極限の待ち時間と待ち行列の長さの期待値を示している．しかし，二つの基本的なプロセッサ共有待ち行列に対する待ち時間の分布は，Schrage [40]とCoffmanら[44]によって得られており，ラウンドロビン系の待ち時間の分布については，最近Muntz [45]によって得られている．計算機スケジューリングの方法の歴史について，分り易くより詳しい概観と参考文献一覧をMcKinney [46]が与えている．タイムシェアリング・システムのアルゴリズムについてのより新しい進歩や拡張の多くは，Adiri等[47～49]によって調べられている．

多くの優先権待ち行列で用いられる横取り割込み可のスケジューリング技法の制約は，横取り割込みの費用である．特に，横取り割込みによるオーバヘッドが使用可能なプロセッサ時間の全てを消費してしまい得るので，これらのシ

† **訳注**　オーバヘッド(overhead)という語は，利用者プログラムに対する直接的な寄与はないけれどもシステム全体の運行には必要な作業に要する経費(とくに計算時間，時には記憶領域等も)を指して使われる．各種の管理プログラムの走行に要する経費と解釈してよい．

ステムでは極端に短い時間量子を用いるのは好ましくない．横取り割込みの費用を考慮に入れた最適スケジューリングについての理論的な記述で，この本を書くまでに発表されたものはない．Mullery と Driscoll [50] によるある種の予備的な研究は，ラウンドロビンの技法では，与えられた時間量子の大きさに対し起り得る横取り割込みの費用が最高になること，およびより低い横取り割込み費用とより望ましい応答時間の分布を持つ他の技法が存在することを示している．

## 1.4 記憶領域管理

電子計算機の最も初期の頃から，計算機の記憶システムはいつも，少なくとも2階層に編成されてきており，**記憶領域管理問題**(memory management problem)——すなわち，それらの階層間にどのように情報を配置し，いつそれを移動させるかを決定する問題——は，いつも一番重要な問題であった．記憶管理方策の目的は，参照される見込みが最も大きいプログラムの部分を主記憶に保持することによって，可能な実行速度を最高にすることである．記憶管理方策は三つの副方策から成っている [51, 52].

1. **取込み時選択の方策**(fetch policy)は，ブロックを補助記憶から主記憶へ前もって移しておくのか，それとも必要時に移すのかということを決定する．
2. **格納位置選択の方策**(placement policy)は，主記憶上まだ割当てられていない領域のうちのどこに，新しく来たブロックを置くかを決定する．
3. **排除ページ選択の方策**(replacement policy)は，主記憶からどのブロックをいつ追い出すかを決定する．

**図1.4-1** は，2階層の記憶システムの構造を示している．この記憶システムは(主記憶と補助記憶との間を指定したブロックを移動させるための)**移送コマンド**(move command)を用意しており，これを用いて三つの副方策を実現することができる．最も初期の計算機では，プログラマはプログラムの命令やデータをブロックに編成し，適当な移送コマンドをそのプログラムの文中に挿入することによって，記憶管理の問題を自分自身の手で解決しなければならなかった．この手法は，記憶管理問題に対する**オーバレイ解法**(overlay solution)として知られている．

1950年代後半以来の高級プログラミング言語の導入により，プログラマはずっと複雑なプログラムを作成できるようになったが，プログラムがより複雑に

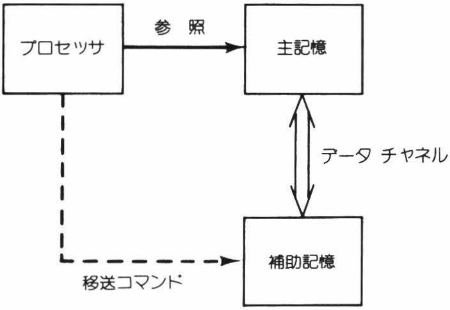



図1.4-1　2階層記憶システム

なってくるのにつれて，オーバレイ問題の重要度も大きくなってきた．(Sayreは，オーバレイ問題の解決のために，プログラミングの費用の25～40%が必要であると評価した[53])．それによって，オーバレイ問題を解決する自動的な手続きの探求が動機づけられたのである．

オーバレイ問題をプログラマから救済する一組の技法は，プログラムのコンパイラを中心としたものであった．これらの技法に従うと，コンパイラはプログラムの構造を解析してブロックに編成し，プログラムの文中に適当な移送コマンドを挿入する．この接近法に対して，初期には相当関心が持たれたのであるが，さらに発展するということはなく，関心は動的記憶領域割当て(dynamic storage allocation)技法や仮想記憶(virtual memory)†技法に転じた．1961年の記憶割当てに関するACMシンポジウムの論文集[54]や1966年のRamamoorthyの論文[55]は，オーバレイ問題に対してコンパイラが作り出す解法について，大変良くまとめられている．Knuthは動的な記憶領域割当てに対するさまざまの重要な技法について優れた議論を行っている[56. pp, 435ff]．

本書では，記憶管理のための仮想記憶技法の解析に注意の焦点を合わせる．これは次の三つの理由による．第一に，1969年にSayreが，仮想記憶技法はオーバレイ技法に対して競争力があり，全体の費用からみると潜在的には秀れてさえいるということを示す重要な論文を発表したこと．第二に，仮想記憶技法は実現する場合の単純さから大いに興味をひくものであり，広く普及するようになってきたこと．第三に，仮想記憶の技法は記憶管理における最も挑戦的で興味深く，かつ一般的な問題を提供し，その解析は，動的な記憶割当ての手法

† **訳注**　最近では，virtual storageという術語が使われる．

の多くの局面に適用することができること.

仮想記憶は，シミュレートされた(または仮想の)計算機の主記憶と見なすことができる．そして，番地空間(または名前空間)，記憶空間(または位置空間)および番地写像という三つの抽象概念で記述される．タスクの**番地空間**(address space)は，プロセッサがそのタスクを実行するときにプロセッサによって生成され得る番地の集合である．大抵のシステムは二つ以上の番地空間を設定できるように設計されている(たとえば多重プログラミング・システム)．あるタスクの情報を，他のタスクがこれに到達することから保護することが望ましいとき，各々のタスクは自分用の番地空間を持つのが普通であり，いくつかのタスクが情報を共有し得ることが望ましいとき，それらは同じ番地空間を共有しても良い．システムの**記憶空間**(memory space)は，物理的な存在としての主記憶内の記憶位置の集合である．**番地写像**(address map)は番地空間 $N$ から記憶空間 $M$ の中への(部分)関数 $f$: $N \rightarrow M$ であり，もし番地 $x$ を持つ $N$ の項目が，与えられた時点において主記憶の位置 $y$ に蓄えられているならば $f(x)$ はその時点で $y$ と定義され，そうでないとき $f(x)$ は未定義である．$f(x)$ が未定義であるような $x$ に対してプロセッサによる参照が試みられると，動作例外の条件が発生しそのタスクへの割込みが引起される．動作例外ハンドラは，この主記憶上に見付からない項目が主記憶上に置かれるように手筈を整え(他のある項目はその過程で主記憶から押しのけられるかも知れない)，番地写像 $f$ を更新し，中断したタスクを再開する.

番地写像の物理的な解釈を**図 1.4-2** に示す．プロセッサは $N$ 内の番地だけを生成する．これらの番地は“写像器(mapper)”に受取られ，写像器は $f$ が記述されている表を調べに行く．もしも $f(x)$ が定義されていて $y$ に等しければ写像器は $M$ に番地 $y$ を提示する．そうでなければ，写像器は適当な移送コマンドを開始し，$f$ を記述している表を修正しておく.

プロセスがプロセッサの一つの抽象概念であるように，番地空間が主記憶の一つの抽象概念であることに注意することが大事である．番地空間を十分大きくすると，その中ではもはやオーバレイを行う必要がないことを保証できるから，オーバレイの問題は解決される．さらに $N$ は，$M$ について前もって設定された諸仮定に依存しないので，タスクは $M$ の大きさの如何にかかわらず実行できる．考察されるべき重要な解析問題の中には，取込み時，格納位置および排除ページ選択の各方策の選び方や性能に関する記憶容量の影響などがある．仮想記憶システムに対する動機や仮想記憶システムの限界についての完全な議論が Denning によって与えられている [1, 52].

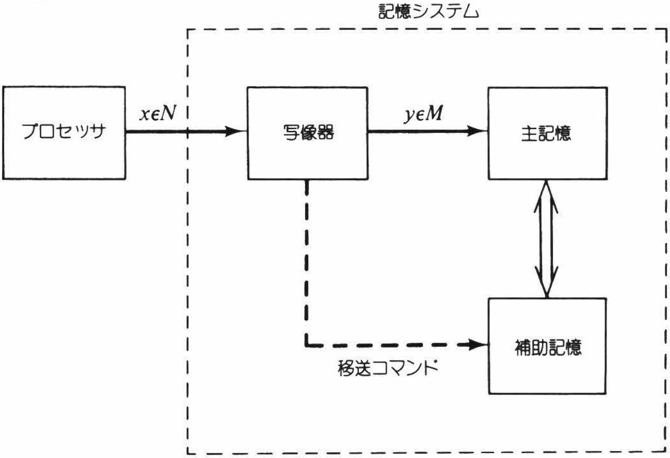



**図 1.4-2**　仮想記憶システム

**図 1.4-3** に示されるように，もしプロセッサと記憶システムとの間の通信路を監視すると，タスクが番地の系列，すなわち**番地軌跡**(address trace) $x_1x_2\cdots x_t\cdots$（ここで，$t\geq 1$ に対し，$x_t\in N$ は時刻 $t$ で参照される番地であり，時間は参照に関しては離散的に測定される）を生成していることが観察されよう．$N$ に属する各 $x$ ごとに $f(x)$ を記入するような表 $f$ は耐え難い程の大きさになるから，1.2 節でも注意したように，大抵の仮想記憶用の補助記憶装置はブロック向きである．ブロック向きの写像では，$f(b)$ は，定義されているならば $M$ におけるブロック $b$ の基底(base)番地を表している．ここで，ある番地 $x$ は，$x$ を含むブロックの番号 $b$ と $b$ の基底に対する $x$ の相対番地 $w$ から成る対 $(b, w)$ に変換される．写像器は $M$ に提示するために $y=f(b)+w$ を計算する．ブロックは情報の割当てと転送の単位であるから，番地軌跡（の概念）は解析に必要なもの以上に詳し過ぎる．そこで，番地 $x_t$ を含むブロックを $r_t$ とするとき，タスクの**参照列**(reference string) $r_1r_2\cdots r_t\cdots$，$t\geq 1$，が定義される．記憶管理方策の解析では，プログラムの振舞のモデルの基礎として，この参照列を用いる．

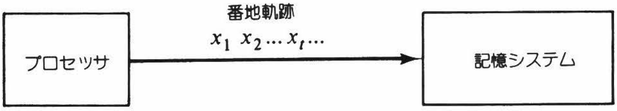



**図 1.4-3**　番 地 軌 跡

記憶管理についての全ての結果は，**ページング**(paging)，すなわち全てのブロックが同じ大きさに揃っている場合について示されている．各ページが $p$ 語から成っているなら，$N$ の番地 $x$ に対応する(ブロック，番号)対は，

$$(b, w) = \left(\left[\frac{x}{p}\right], \; x \bmod p\right)$$

で与えられる．ここで $[x/p]$ は，$x/p$ を越えない最大の整数を表し，$x \bmod p$ は$p$ を法とする $x$ の剰余を表す．大抵の計算機では，番地計算は，2進の算術演算およびある $q>0$ に対して $p=2^q$ であることを用いて行われ，その結果 $x$ からの$(b, w)$の計算は自明となっている(図1.4-4)．我々はページ化された記憶システムを仮定しているから，参照列はページ番号の系列となる．ページングに限定することは明らかに解析の一般性をいくらか犠牲にする．それにもかかわらず，ブロックの大きさの確率分布を与えてブロックの大きさを可変にして解析をやり直したとしても，多くの結果は大して変ったものにはならないということが分るであろう．

全ての仮想記憶システムは1960年以前に Atlas 計算機で使われた"1階層記憶†(one-level store)" [57] から派生したものであり，この Atlas はページングの計算機であったのである．ページングの技法は今や広く使われており，必要時ページング(demand paging)の技法(この場合取込み時選択の方策は純粋に必要時のものである)は実現が非常に簡単であることから特にそうである [52].

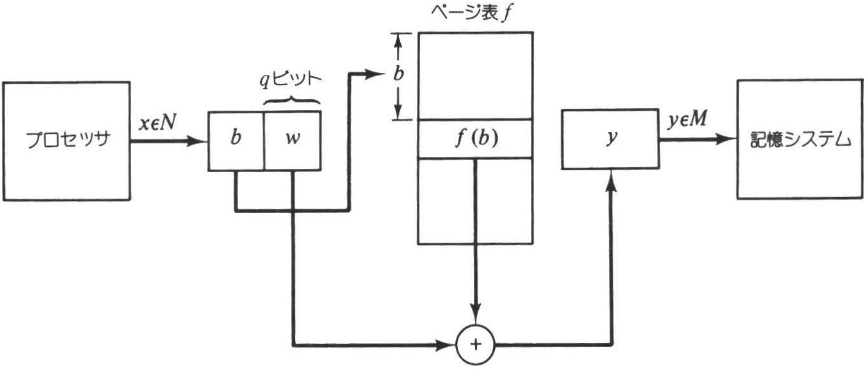



**図1.4-4** ページングにおける番地の形成

† **訳注** Atlas のハードウェアは磁心記憶と磁気ドラムの2階層になっているが，利用者プログラムから見れば1階層になっているという意味である．

ブロックの大きさが可変である場合の仮想記憶の技法(セグメンテーション(segmentation)方式として知られている)は，いくつかの計算機，とりわけBurroughs 社の B5000 やそれに続くシリーズで成功裡に使われてきている．セグメンテーションとページングの組合わせは Dennis の有名な論文 [58] に記されているように，ほんのいくつかの計算機，とくに Multics [59] で実現されている．これらの三つの技法——ページング，セグメンテーション，その両方——の比較は Denning によって与えられている [52].

仮想記憶を実現するときは大抵，タスクの写像表を主記憶に蓄える．したがって名前空間の番地から記憶位置を計算するのに，写像表を調べに行く必要から2倍ないしそれ以上(番地写像を構成する表のレベルの数に依存する)の潜在的な速度の低下がある．それゆえ，大抵の仮想記憶の写像器には，最も最近に使われた写像表の項目を含む，小さな連想記憶(associative memory)が組込まれている．番地の形成に際して，与えられた番地 $x$ は上述のように対$(b, w)$に変換される．もしある連想記憶用レジスタが項$(b, f(b))$を内容として持っていると，記憶番地 $y = f(b) + w$ は，表 $f$ への余分な到達なしに直ちに計算することができる．(もちろん，もしそのような項が連想記憶内になければ $f$ への余分な到達が必要となろう.) Schroeder は，3階層の写像表を用いている Multics では，16 セルの連想記憶を用いると，全ての写像表の項が0時間で到達できる場合と比べた速度の低下は，3.5% を越えないということを報告している[60]．このように，仮想記憶の性能は，取込み時，格納位置と排除ページ選択の3方策によって左右されるのであって，写像の機構によってではないということが分る．

### 1.4.1 補助記憶と緩衝記憶の問題

第5章では補助記憶についていくつかの重要な問題を取扱う．大抵の補助記憶装置の到達時間は，システム内の他の記憶装置の到達時間よりも相当に長いから，これらの装置がシステム内の輻輳点になる傾向がある．したがって，これらの装置を管理する場合の方針は，主としてそれらの処理率(throughput，単位時間当りの仕事の処理量)を最大にすることに関係している．本書ではこれらの問題を，その解析の手法が第4章で開発した確率モデルの直接の応用であることから，主記憶管理の問題よりも先に取扱う．

第5章で考える最初の問題はページング・ドラム(図 1.2-2 参照)の解析であるが，このドラムはページ化された記憶システムでは広く使われている．もし，ドラム使用の要求が，セクタごとの待ち行列に区分されており，かつ，ドラム

が，ヘッドの下を通過しているセクタに対応している待ち行列の先頭の要求に対してサービスをするならば，そのドラムは**最短到達時間順**(shortest-access-time-first, SATF)方策に従って振舞うことになろう．SATF 方策は，総計の累積回転待ち時間を最小にするから，ドラムの処理率を最大にするという意味で最適である．実際，セクタ数 $n$ のページング・ドラムは，全ての要求に対して一本の FIFO 型待ち行列を用いるドラム・システムよりも，おおよそ $n/2$ 倍も大きい処理率を持つ．ページング・ドラムは，1960 年代の初期から商業的に利用可能であったが，それを解析する最初の試みは，1960 年代中葉まで完成しなかった．Denning はそれを 1965 年に [61]，Weingarten は 1966 年に [62] 論じ，Coffman は 1969 年にその振舞についての完全な解析を初めて示した [63]．それに続く 1969 年の Abate と Dubner [64]，1971 年の Burge と Konheim [65]，1971 年の Greenberg [66]，1972 年の Denning [67] 達の研究は，このドラム・システムについて分り易い知識を与えてくれた．

可動アーム・ディスク(図 1.2-4 参照)の振舞を解析する試みは余り成功したようには見えない．問題は SATF スケジューリング方策——実際は**最短探索時間順**(shortest-seek-time-first, SSTF)方策になるが——では，一番内側のシリンダや一番外側のシリンダへの要求を冷遇しがちになることである．全体でみた処理率は最大になるかも知れないが，内側や外側のシリンダに対する要求は余りにも貧弱なサービスしか受けられないのでシステム全体(そのディスク装置はこのシステムの構成要素である)の性能は，ひどく悪くなる．1967 年に Denning は SCAN と呼ばれるアーム制御の方策を提案した[68]が，これは与えられたアームの移動方向について SSTF 方策を使うようにするもので，アームは与えられた方向の前方には要求がもはやなくなるまで，移動方向を反転することを許されない．この方策は，SSTF 方策と，差別待遇のない(しかし低い処理率の)FIFO 方策との間の現実的な折衷案となっている．第 5 章には，その解析が記されている．その他の解析として，1966 年の Seaman 等によるもの [69] や，ディスク・スケジューリングの既知の方策についての広範囲の研究である 1972 年の Teorey と Pinkerton によるもの [70] も記されている．彼等の結果は，SCAN がうまく振舞うことを示している．さらに，一つの方策で，負荷条件の全範囲にわたって適合するようなものはないことも示している．

補助記憶や入出力装置(遠隔地のものもそうでないものも)と主記憶との間の転送は全て緩衝記憶を使うから，それに関連したある種の緩衝記憶の問題の解析も行っている．その結果は，各プロセスに一つずつ別々の緩衝記憶を持たせ

るよりも，緩衝記憶を共用化した方が(記憶媒体の利用度に関しては)かなり効率が良くなることを示している．共用化された緩衝記憶に対してさらに解析し，入出力プロセスのパラメータと要求される緩衝記憶の大きさの分布から，緩衝記憶の占有や溢れの確率を与えている．主記憶は多重プログラミングの下では(共有化された)緩衝記憶と見なすことができるから，これらの結果は多重プログラミング下の記憶管理の問題に直ちに応用できる．緩衝記憶の問題についての最も初期の解析の中に，可変長のメッセージが何個かの固定長のブロックを使って蓄えられる場合に緩衝記憶領域として最適のセルの大きさを論じた1965年の Wolman の研究がある [71]．これらの結果は，可変長のセグメントやファイルがページの集合として蓄えられなければならない場合のページの大きさの最適化問題にある程度まで適用できる．共有化した緩衝領域と専用緩衝領域との比較についての初期の研究としては1968年の Denning によるものがある [72]．最も重要な研究は1971年に Gaver と Lewis によって行われているが [73]，これが本書でこの問題を取扱う動機となったのである．1972年，Coffman と Ryan は同様の手法を多重プログラミングに応用した [74]．

これまでに扱った全ての待ち行列モデルと確率モデルは，環境としての役割を果している外界の計算機システムの振舞には無関係に，特定のサブシステムの振舞に対する洞察を与える．これらのモデルはそれぞれ輻輳点が一つ，すなわち一つの扱い者(サーバ，server)を含んだものである．これらのモデルでは，普通は多くの輻輳点を含む計算機システム全体の振舞は予測しかねる．このため，Jackson [75] や，Gordon と Newell [76] による重要な結果を記しておいたが，これはネットワーク内の各扱い者のサービス時間が指数分布に従っており，かつネットワーク内で活動中のタスクの総数が一定であるという仮定の下に，任意の閉じたネットワークの解析を考察したものである．これらの結果は，将来特別に重要な意味をもつようになると信じているが1971年以来の Baskett [77, 78] や Kimbleton と Moore [78, 80] による研究では，各扱い者が同じ平均サービス時間を持つような指数的扱い者で置換えられた待ち行列ネットワークは，もとの(実世界の)システムの振舞を意外な程良い近似度で予測することを示している．これらの結果は，システム内の個々の扱い者の詳細な振舞よりもむしろモデルが本当のシステムの構造を反映することの方が遥かに重要であるということを示唆している．終りに，1971年の Buzen の研究 [81] や Chandy の最近の研究 [107] は，そのようなネットワーク内の流れを最適化する問題として資源の割当てを論じたものであるが，一般の資源割当て問題への有益な洞察をもたらすことを約束しているといえよう．

### 1.4.2 ページング・アルゴリズム

第6章では主記憶管理の問題に注意を向けるが，この問題は通常はプログラムの最大実行速度に関連している．ページングに注意を限ると，格納位置選択の方策が性能に及ぼす効果を無視できて解析を簡単化できる．さらに注意の大部分を必要時ページングに限ると，取込み時選択の方策が性能に及ぼす効果も無視できる(これらの二つの制限の下では，記憶管理方策の性能は排除ページ選択の方策に支配される)．ページング，特に必要時ページングは広く使われているし，可変長ブロックに迎合した解析を行ったとしても多くの結果は質的には変りがないので，これらの制限は必ずしも厳しいものではない．

第6章の結果は，ページ枠が $m$ 個ある記憶上において必要時ページングで動作する $n$ ページの(参照列で記述された)プログラムが一つだけ動作する場合を扱ったものである($1 \leq m \leq n$)．ページの大きさやページ間の情報の分布についての論点はこの問題の中で示されている．多重プログラミングおよびプログラムに可変量の空間を割当てることに関連した問題は第7章に譲る．

ページング・アルゴリズムの振舞の二つの局面が第6章で論じられる．第1の局面は，最適ページング・アルゴリズムの問題と最適の排除ページ選択の原理である．費用関数とはあるプログラムの実行の過程を通じて(主)記憶に移されるページ数であると評価するものとすれば，あらゆるページング・アルゴリズムは必要時ページングに変換することによって改善されるということが示される．この結果は，Mattson 等が1970年の良く知られた論文 [82] で明らかにしたものである．ページングの方策でプログラムの参照列を先読みすることが許される場合は，最適な必要時ページング方策は最も長い間再使用されないページを排除するということも示される．この方策は1966年に Belady [83] によって MIN アルゴリズムとして提案されたが，それが最適であることの証明は1970年に Mattson 等 [82] と Aho 等 [83] によって独立に示された(このアルゴリズムの動作原理は，1966年に Horwitz 等[85]が提案したインデックス・レジスタの割当てアルゴリズムに良く似ている)．ページングの方策が将来の先読みを許されていないとき，“最も長い間，再使用されることが期待されないページを排除する”という規則は良い発見的手法であり，ページ参照列が独立した確率変数の定常的な系列として扱われる独立参照モデル(independent-reference model) [84] や，いわゆる LRU-スタック・モデル(LRU-stack model) [74, 82, 86] のように，この規則が最適な方策を生むことが知られている場合もある．

第6章で扱われる第2の局面は，必要時ページングのアルゴリズムと等価な

二つの重要なクラスである**優先権アルゴリズム**(priority algorithm)と**スタック・アルゴリズム**(stack algorithm)の振舞に関するものである．これらのアルゴリズムは1970年にMattson等[82]によって提案された．実用上興味のある大抵のページング方策は，規則が“与えられた優先権リストに従って最低の優先権のページを排除する”という方策を含むスタック・アルゴリズムである．これらのアルゴリズムが重要であるのは，これらの振舞が良性の(すなわち，変則的なことがない)ことと解析が比較的し易いことによる．与えられた大きさの(主)記憶で，スタック・アルゴリズムによって生成された(取込み，排除)の対の系列だけが与えられたとき，任意に大きい(主)記憶での(取込み，排除)の対の系列を予測することができるということも示される．これは**拡張問題**(extension problem)として知られており，CoffmanとRandellによって1971年に初めて考察された[87]．

### 1.4.3　プログラムの振舞と多重プログラミング

第7章には多重プログラミング下の記憶システムにおける最適な記憶管理に関連した解析が示されている．第6章の解析の多くについては，多重プログラミングへと拡張できる自然な方法がない．その主な理由は，多重プログラミングの戦略は参照列と振舞についての確率的な仮定に対してのみ影響を与えるように樹てられるのに対して，第6章の解析は与えられた参照列に対する与えられたページング・アルゴリズムの振舞を取扱っていることによる．

プログラムの振舞の基本的な性質としての**局所性**(locality)を考える．任意の時間間隔の間，プログラムのページの参照のされ方が一様でなく，かつページの参照の密度が時間と共に徐々にしか変化しないとき，参照列はこの性質を満たす．過去の参照のパターンに基づいて将来の参照のパターンを予測することは，ページの参照について強い局所性を示すプログラム程，確かなものになることは明らかである．局所性の性質は長い間プログラミングの際に観察されてきたが，この性質は，問題の他の部分に注意を切換える前に，しばらくの間は大きな問題の小さな部分に注意を集中するという，プログラマの自然な傾向によるものと思われる．この性質はまた，プログラムの振舞についての経験的な研究でも一貫して観察されているが，最も良く知られている研究として，1966年のFine等[88]，1966年のBelady[83]，1968年のCoffmanとVarian[89]，1970年のKuckとLawrie[90]によるものがある．1968年と1970年のBrawnとGustavsonによる一対の研究[91, 92]では，ページングの仕組みは，プログラマがほんのわずかな努力をするだけで，最も注意深く設計され

た手作りのオーバレイの仕組みと性能上，比肩できる程著しくプログラムの局所性を改善できることを立証したが，この結論は1969年にSayreによって確証され強調された[53]．局所性に従ったプログラムの性質を論じたその他の研究には，BeladyとKuehnerによる1969年の空間の動的な共有を扱ったもの[93]，McKellarとCoffmanによる局所性をできるだけ高めるための配列の構成と配列操作の手続きに関する解説[94]や，局所性を高くするためにコードをページ間に配分する方法を示したKernighanの1970年の解析[95]などがある．

任意に与えられた時点において，タスクの処理効率が指定された水準以上になることを保証するために主記憶内になければならないようなページの最小の部分集合を，そのタスクの作業集合(working set)と定義できる．あるタスクの作業集合——過去に向けて，後方に開かれた与えられた大きさの窓に観察されたページの集合として——を測定する実際的な手法は1968年のDenningの論文で提案された[96]．あるタスクの近い将来の記憶の需要を予測する道具としてこの測定法が有効であるのは局所性による．すなわち，ある与えられた時点の作業集合内のどのページも，近い将来も多分そのまま作業集合内に留まっているからである．記憶管理の**作業集合原理**(working set principle)は，タスクはその作業集合が主記憶内に存在するときのみ“処理可能”であるといわれることを主張している．この原理の有効さは，Multics, IBMのTSS/360[97]，PDP-10を使ったBBNのTENEXシステム[98]やRCAのSpectra 70/46 [99, 100] などのシステムで実際に証明されている．この原理は，スラッシング(thrashing)(すなわち，作業集合(の大きさ)に比べて割当てられた主記憶が少な過ぎることが誘因となってページの入換えが頻発し，性能が極度に低下する現象)の問題を制御するために示されてきた[52, 101]．第7章には，局所性をモデル化する手法や，局所性を持つ参照列に対する作業集合モデルの解析を記す．その解析の結果として，作業集合の大きさ，ページング(入出力)が生じる割合，ページが参照される時間間隔の分布と**最終参照時刻順**(least-recently used, LRU)のページング技法を用いた作業集合の記憶管理を実現する方法との関係に関するものがある．これらの解析は，DenningとSchwartzによって1972年に報告された研究[102]の道筋に沿ったもので，その結果は1972年にRodriguez-Rosellによって確証されている[103]．1972年，Denning 等は局所性を具体的に表現するプログラムのモデルを仮定し，このモデルに対して作業集合モデルが最適に近い記憶管理の戦略になっていることを証明した[86]．

局所性の仮定をおくと，タスクの作業集合の大きさは正規分布になる傾向があることを証明できる．局所性に従うそれらのタスクに対してこのことは重要な結果である．なぜならば，正規過程に関する膨大な文献が記憶管理の研究に応用できることを意味するからである(タスクの記憶領域に対する需要は正規過程であると考えられている)．この結果は Denning と Schwartz によって報告され [102]，Coffman と Ryan の実験 [74] と一致している．Rodriguez-Rossel は局所性に従わないプログラムの例を示し，それらの作業集合は，正規過程になる傾向がないことをも示した [103]．

タスク $T_i(i=1, 2, \cdots, n)$ の番地空間を単一の記憶空間 $M$ の中へ写像したいものと仮定する．**固定分割**(fixed-partition)の戦略に従うと，各タスク $T_i$ は $M$ の静的分割の1ブロックを割当てられ，あるページング・アルゴリズムが，任意に与えられた時点でその番地空間のどの部分集合がそのブロックに入るかを決定する．**可変分割**(variable-partition)の戦略に従うと，各タスク $T_i$ は，与えられた時点で，その時の作業集合を保持するに足る量の空間を割当てられる．第5章の緩衝記憶の解析を一部変更すると，時間の関数としてのタスクの作業集合が正規過程であるという仮定の下に，**溢れの確率**(overflow probability)と，**溢れの総量の期待値**(expected total amount of overflow)という尺度に関しては上の二つの戦略の性能を比較することができる．典型的なパラメータの値に対し，可変分割の戦略(作業集合原理)の方が決定的に優れている．これらの結果は1972年に Coffman と Ryan が示した結果 [74] から導かれる．

## 参考文献


1. Denning, P. J., "Third generation computer systems." *Computing Surveys 3*, 4 (Dec. 1971), 175–216.
2. Rosen, S., "Electronic computers: An historical survey." *Computing Surveys 1*, 1 (Mar. 1969), 7–36.
3. Holt, R. C., "On deadlock in computer systems." Ph.D. thesis, Dept. of Computer Science, Cornell Univ., Ithaca, N.Y., TR–71–91 (1971).
4. Dennis, J. B., and E. G. Van Horn, "Programming semantics for multiprogrammed computations." *Comm. ACM 9*, 3 (Mar. 1966), 143–155.
5. Saltzer, J. H., "Traffic control in a multiplexed computer system." M.I.T. Project MAC report MAC–TR–30, M.I.T., Cambridge, Mass. (1966).
6. Van Horn, E. C., "Computer design for asynchronously reproducible multiprocessing." M.I.T. Project MAC report MAC–TR–34, M.I.T., Cambridge,

Mass. (1966).

7. Horning, J. J., and B. Randell, "Structuring complex processes." IBM, T. J. Watson Research Center report RC-2459, Yorktown Heights, N.Y. (May 1969).
8. Muller, D. E., and W. S. Bartky, "A theory of asynchronous circuits." *Proc. Int'l Symp. Thy. Sw., An. Comp. Lab. Harv. U. 29*, Pt I (1959), 204–243.
9. ACM. Record of the project MAC conference on concurrent systems and parallel computation. Wood's Hole, Mass. (June 1970).
10. Bernstein, A. J., "Analysis of programs for parallel programming." *IEEE Trans. Comp. EC-15*, 5 (1966), 757–762.
11. Dijkstra, E. W., "Cooperating sequential processes," in *Programming Languages* (F. Genuys, ed.). Academic Press, New York (1968), 43–112.
12. Dijkstra, E. W., "Solution of a problem in concurrent programming control." *Comm. ACM 8*, 9 (Sept. 1965), 569.
13. Habermann, N., "Prevention of system deadlock." *Comm. ACM 12*, 7 (July 1969), 373–377.
14. Havender, J. W., "Avoiding deadlock in multitasking systems." *IBM Sys. J. 7*, 1 (1968), 74–84.
15. Murphy, J., "Resource allocation with interlock detection in a multitask system." *AFIPS Conf. Proc. 33* (1968 FJCC), 1169–1176.
16. Petri, C. A., "Communication with automata." Suppl. 1 to RADC-TR-65-377 (Vol. 1) Griffis AFB, Rome, N.Y. (1966). [Original in German, Univ. of Bonn (1962).]
17. Karp, R. M., and R. E. Miller, "Parallel program schemata." *J. Comp. Sys. Sci. 3*, 2 (May 1969), 147–195.
18. Luconi, F., "Asynchronous control structures." M.I.T. Project MAC report MAC-TR-49, M.I.T., Cambridge, Mass. (1968).
19. Slutz, D. R., "The flow graph schemata model of parallel computation." M.I.T. Project MAC report MAC-TR-53, M.I.T., Cambridge, Mass. (1968).
20. Rodriguez, J., "A graph model for parallel computation." M.I.T. Project MAC report MAC-TR-64, M.I.T., Cambridge, Mass. (1969).
21. Holt, A. W., and F. Commoner, "Events and conditions." *Rec. Pr. MAC Conf. on Concur. Sys. Par. Comp.* (June 1970) [9].
22. Johnson, S. M., "Optimal two and three stage production schedules with set-up times included." *Nav. Res. Log. Quart. 1*, 1 (Mar. 1954).
23. Clark, W., *The Gantt Chart* (3rd ed.). Sir Isaac Pitman & Sons Ltd., London

(1952).

24. Moder, J. J., and C. R. Phillips, *Project Management with CPM and PERT*. Van Nostrand Reinhold, New York (1964).

25. McNaughton, R., "Scheduling with deadline and loss functions." *Manag. Sci. 12*, 1 (Oct. 1959).

26. Hu, T. C., "Parallel sequencing and assembly line problems." *Opns. Res. 9*, 6 (Nov. 1961), 841–848.

27. Manacher, G. K., "Production and stabilization of real-time task schedules." *J. ACM 14*, 3 (July 1967), 439–465.

28. Graham, R. L., "Bounds on multiprocessing time anomalies." *SIAM J. Appl. Math. 17*, 2 (Mar. 1969), 416–429.

29. Fujii, M., T. Kasami, and K. Ninomiya, "Optimal sequencing of two equivalent processors." *SIAM J. Appl. Math. 17*, 4 (July 1969), 784–789.

30. Coffman, E. G., Jr., and R. L. Graham, "Optimal scheduling for two-processor systems." *Acta Informatica 1*, 3 (1972), 200–213.

31. Muntz, R. R., and E. G. Coffman, Jr., "Preemptive scheduling of real-time tasks on multiprocessor systems." *J. ACM 17*, 2 (Apr. 1970), 324–338.

32. Conway, R. W., W. L. Maxwell, and L. W. Miller, *Theory of Scheduling*. Addison-Wesley, Reading, Mass. (1967).

33. Krone, M., "Heuristic programming applied to scheduling problems." Ph.D. thesis, Dept. of Electrical Engineering, Princeton Univ., Princeton, N.J. (Sept. 1970).

34. Lampson, B. W., "A scheduling philosophy for multiprocessing systems." *Comm. ACM 11*, 5 (May 1968), 347–359.

35. Coffman, E. G., Jr., and L. Kleinrock, "Computer scheduling methods and their countermeasures." *AFIPS Conf. Proc. 32* (1968 SJCC), 11–21.

36. Corbato, F. J., M. M. Daggett, and R. C. Daley, "An experimental time sharing system." *AFIPS Conf. Proc. 21* (1962 FJCC), 335.

37. Kleinrock, L., "Analysis of a time shared processor." *Nav. Res. Log. Quart. 11*, 10 (March 1964), 59–73.

38. Coffman, E. G., Jr., and B. Krishnamoorthi, "Preliminary analyses of time-shared computer operation." Technical report SP-1719, Systems Development Corp., Santa Monica, Calif. (Aug. 1964).

39. Krishnamoorthi, B., and R. C. Wood, "Time-shared computer operations with both interarrival and service time exponential." *J. ACM 13*, 3 (July 1966), 317–338.

40. Schrage, L. E., "The queue M/G/1 with feedback to lower priority queues."

*Manag. Sci. 13* (1967), 466–474.

41. Coffman, E. G., Jr., and L. Kleinrock, "Feedback queueing models for time-shared systems." *J. ACM 15*, 4 (Oct. 1968), 549–576.

42. Scherr, A. L., *An Analysis of Time-Shared Computer Systems.* M.I.T. Press, Cambridge, Mass. (1966).

43. Cobham, A., Priority assignment in waiting line problems." *Opns. Res. 2* (Feb. 1954), 70–76.

44. Coffman, E. G., Jr., R. R. Muntz, and H. Trotter, "Waiting time distributions for processor sharing systems." *J. ACM 17*, 1 (Jan. 1970), 123–130.

45. Muntz, R. R., "Waiting time distribution for round-robin queueing systems." *Proc. Symp. Comp.-Commun., Networks, Teletraffic*, Microwave Research Institute, Polytechnic Institute of Brooklyn (Apr. 1972).

46. McKinney, J. M., "A survey of analytical time sharing models." *Computing Surveys 1*, 2 (June 1969), 105–116.

47. Adiri, I., and B. Avi-Itzhak, "A time-sharing queue." *Manag. Sci. 15*, 11 (1969), 639–657.

48. Adiri, I., "A dynamic time-sharing queue." *J. ACM 18*, 4 (Oct. 1971), 603–610.

49. Adiri, I., "A note on some mathematical models of time-sharing systems." *J. ACM 18*, 4 (Oct. 1971), 611–615.

50. Mullery, A. P., and G. C. Driscoll, "A processor allocation method for time-sharing." *Comm. ACM 13*, 1 (Jan. 1970), 10–14.

51. Randell, B., and C. Kuehner, "Dynamic storage allocation systems." *Comm. ACM 11*, 5 (May 1968), 297–306.

52. Denning, P. J., "Virtual memory." *Computing Surveys 2*, 3 (Sept. 1970), 153–189.

53. Sayre, D., "Is automatic folding of programs efficient enough to displace manual?" *Comm. ACM 12*, 12 (Dec. 1969), 656–660.

54. ACM. Proceedings of a symposium on storage allocation. *Comm. ACM 4*, 10 (Oct. 1961).

55. Ramamoorthy, C., "The analytic design of a dynamic look ahead and program segmentation system for multiprogrammed computers." *Proc. 21st Nat'l ACM Conf.* (1966), 229–239.

56. Knuth, D. E., *The Art of Computer Programming* (Vol. 1). Addison-Wesley, Reading, Mass. (1968).

57. Kilburn, T., D. B. G. Edwards, M. J. Lanigan, and F. H. Sumner, "One-

level storage system." *IRE Trans. EC-11* (Apr. 1962), 223–238.

58. DENNIS, J. B., "Segmentation and the design of multiprogrammed computer systems." *J. ACM 12*, 4 (Oct. 1965), 589–602.

59. BENSOUSSAN, A., C. T. CLINGEN, and R. C. DALEY, "The Multics virtual memory." *Proc. 2nd ACM Symp. Op. Sys. Princ.* (Oct. 1969), 30–42.

60. SCHROEDER, M. D., "Performance of the GE-645 associative memory while Multics is in operation." *Proc. ACM Symp. Sys. Perf. Eval.* (Apr. 1971), 227–245.

61. DENNING, P. J., "Queueing models for file memory operations." M.I.T. Project MAC report MAC-TR-21, M.I.T., Cambridge, Mass. (Oct. 1965).

62. WEINGARTEN, A., "The Eschenbach drum scheme." *Comm. ACM 9*, 7 (July 1966), 509–512.

63. COFFMAN, E. G., JR., "Analysis of a drum input/output queue under scheduled operation in a paged computer system." *J. ACM 16*, 1 (Jan. 1969), 73–90.

64. ABATE, J., and H. DUBNER, "Optimizing the performance of a drum-like storage." *IEEE Trans. Comp. C-18*, 11 (Nov. 1969), 992–996.

65. BURGE, W. H., and A. G. KONHEIM, "An accessing model." *J. ACM 18*, 3 (July 1971), 400–404.

66. GREENBERG, M. L., "An algorithm for drum storage management in time sharing systems." *Proc. 3rd ACM Symp. Op. Sys. Princ.* (Oct. 1971).

67. DENNING, P. J., "A note on paging drum efficiency." *Computing Surveys 4*, 1 (Mar. 1972), 1–3.

68. DENNING, P. J., "Effects of scheduling on file memory operations." *AFIPS Conf. Proc. 30* (1967 SJCC), 9–21.

69. SEAMAN, P. H., R. A. LIND, and T. L. WILSON, "Analysis of auxiliary storage activity." *IBM Sys. J. 5*, 3 (1966), 158–170.

70. TEOREY, T. J., and T. B. PINKERTON, "A comparative analysis of disk scheduling policies." *Comm. ACM 15*, 3 (Mar. 1972), 177–184.

71. WOLMAN, E., "A fixed optimum cell-size for records of various lengths." *J. ACM 12*, 1 (Jan. 1965), 53–70.

72. DENNING, P. J., "A model for console behavior in multiuser computers." *Comm. ACM 11*, 9 (Sept. 1968), 605–612.

73. GAVER, D. P., JR., and P. A. W. LEWIS, "Probability models for buffer storage allocation problems." *J. ACM 18*, 2 (Apr. 1971), 186–198.

74. COFFMAN, E. G., JR., and T. A. RYAN, "A study of storage partitioning

using a mathematical model of locality." *Comm. ACM 15*, 3 (Mar. 1972), 185–190.

75. JACKSON, J., "Job shop-like queueing systems." *Manag. Sci. 10*, 1 (1963), 131–142.

76. GORDON, W. J., and G. F. NEWELL, "Closed queueing systems with exponential servers." *Opns. Res. 15*, 2 (Apr. 1967), 254–265.

77. BASKETT, F., "Mathematical models of multiprogrammed computer systems." TSN-17, Computation Center, Univ. of Texas at Austin (Jan. 1971).

78. BASKETT, F., "The dependence of computer system queues upon processing time distribution and central processor scheduling." *Proc. 3rd ACM Symp. Op. Sys. Princ.* (Oct. 1971).

79. KIMBLETON, S. R., and C. G. MOORE III, "A limited resource approach to system performance evaluation." Dept. of Industrial Engineering, Univ. of Michigan, Ann Arbor, TR-71-2 (June 1971).

80. MOORE, C. G., III, "Network models for large scale time sharing systems." Dept. of Industrial Engineering, Univ. of Michigan, Ann Arbor, TR-71-1 (Apr. 1971).

81. BUZEN, J., "Analysis of system bottlenecks using a queueing network." *Proc. ACM Symp. Sys. Perf. Eval.* (Apr. 1971), 82–103.

82. MATTSON, R. L., J. GECSEI, D. R. SLUTZ, and I. L. TRAIGER, "Evaluation techniques for storage hierarchies." *IBM Sys. J. 9*, 2 (1970), 78–117.

83. BELADY, L. A., "A study of replacement algorithms for virtual storage computers." *IBM Sys. J. 5*, 2 (1966), 78–101.

84. AHO, A. V., P. J. DENNING, and J. D. ULLMAN, "Principles of optimal page replacement." *J. ACM 18*, 1 (Jan. 1971), 80–93.

85. HORWITZ, L. P., R. M. KARP, R. E. MILLER, and S. WINOGRAD, "Index register allocation." *J. ACM 13*, 1 (Jan. 1966), 43–61.

86. DENNING, P. J., J. E. SAVAGE, and J. R. SPIRN, "Models for locality in program behavior." Dept. of Electrical Engineering, Princeton Univ., Princeton, N.J., Computer Science technical report TR-107 (May 1972).

87. COFFMAN, E. G., JR., and B. RANDELL, "Performance predictions for extended paged memories." *Acta Informatica 1*, 1 (1971), 1–13.

88. FINE, G. H., C. W. JACKSON, and P. V. MCISSAC, "Dynamic program behavior under paging." *Proc. 21st Nat'l. ACM Conf.* (1966), 223–228.

89. COFFMAN, E. G., JR., and L. C. VARIAN, "Further experimental data on the behavior of programs in a paging environment." *Comm. ACM 11*, 7 (July

1968), 471–474.

90. KUCK, D. J., and D. H. LAWRIE, "The use and performance of memory hierarchies—A Survey," in *Software Engineering* (Vol. I) (J. T. Tou, ed.). Academic Press, New York (1970), 45–78.

91. BRAWN, B., and F. GUSTAVSON, "Program behavior in a paging environment." *AFIPS Conf. Proc. 33* (1968 FJCC), 1019–1032.

92. BRAWN, B., and F. GUSTAVSON, "Sorting in a paging environment." *Comm. ACM 13*, 8 (Aug. 1970), 483–494.

93. BELADY, L. A., and C. J. KUEHNER, "Dynamic space sharing in computer systems." *Comm. ACM 12*, 5 (May 1969), 282–288.

94. MCKELLAR, A. C., and E. G. COFFMAN, JR., "Organizing matrices and matrix operations for paged memory systems." *Comm. ACM 12*, 3 (Mar. 1969), 153–164.

95. KERNIGHAN, B. W., "Optimal sequential partitions of graphs." *J. ACM 18*, 1 (Jan. 1971), 34–40.

96. DENNING, P. J., "The working set model for program behavior." *Comm. ACM 11*, 5 (May 1968), 323–333.

97. DOHERTY, W., "Scheduling TSS/360 for responsiveness." *Proc. AFIPS Conf. 37*, (1970 FJCC), 97–112.

98. BOBROW, D. G., J. D. BURCHFIEL, D. L. MURPHY, and R. S. TOMLINSON, "TENEX—A paged time sharing system for the PDP-10." *Comm. ACM 15*, 3 (Mar. 1972), 135–143.

99. OPPENHEIMER, G., and N. WEIZER, "Resource management for a medium scale time sharing operating system." *Comm. ACM 11*, 5 (May 1968), 313–322.

100. WEIZER, N., and G. OPPENHEIMER, "Virtual memory management in a paging environment." *AFIPS Conf. Proc. 34* (1969 SJCC), 249–256.

101. DENNING, P. J., "Thrashing: Its causes and prevention." *AFIPS Conf. Proc. 33* (1968 FJCC), 915–922.

102. DENNING, P. J., and S. C. SCHWARTZ, "Properties of the working set model." *Comm. ACM 15*, 3 (Mar. 1972), 191–198.

103. RODRIGUEZ-ROSELL, J., "Experimental data on how program behavior affects the choice of scheduler parameters." *Proc. 3rd ACM Symp. Op. Sys. Princ.* (Oct. 1971).

104. KNUTH, DONALD E., "Additional comments on a problem in concurrent programming." *Comm. ACM 9*, 5 (May 1966), 321–323.

105. KARP, RICHARD M., and RAYMOND E. MILLER, "Properties of a model for parallel computations: determinacy, termination, and queueing." *SIAM J. Appl. Math. 14*, 6 (Nov. 1966), 1390–1411.

106. SHOSHANI, A., and E. G. COFFMAN, JR., "Sequencing tasks in multi-process multiple resource systems to avoid deadlocks." *Proc. 11th Symp. Switching and Automata Theory* (Oct. 1970), 225–233.

107. CHANDY, K. M., "Local balance in queueing networks with interdependent servers with applications to computer systems." *Proc. 10th Allerton Conf.*, U. of Ill., Urbana (Oct. 1972).

# 2 並行プロセスの制御

## 2.1 はじめに

並行的に実行しつつあるプログラムは，資源が不適切に共有されていたりプロセス間の結果(信号)の通信が不完全であったりすると，数々の望ましくないあり方で相互に干渉する．一般的にいえば，この問題はいつも，タイミングの問題である．したがって，これに対応して生ずる動作の不首尾は，情報が正しく使われる前に書き直されたり，休止状態にあるべきプロセスが実行されたり，二つのプロセスが互いに相手の進行を待つ“お先にどうぞ(after-you, after-you)”式のループにはまり込む等々の形になって現れる．

本章では順序制御の四つの問題である決定性，閉鎖，相互排除および同期の問題を見極めよう．これらの問題について述べる前に，それを研究するための枠組を与える基本的な定義を示す．まず最初に，十分に一般性のある**プロセス**(process)の定義とプロセスが発生する機構が必要である．

**タスク**(task)は，本章と次章の順序付けの問題(sequencing problem)における計算活動の単位を構成する．タスクはその対外部的な振舞，たとえば，それが要求する入力や生成する出力，その作用または機能とか実行時間などのみによって規定される．タスクの内部操作は何も規定されないし，議論する関心もない．一つのタスクに付随して，**開始**(initiation)と**終了**(termination)という二つの事象がある．$T$ がタスクであるとき，$T$ の開始を $\bar{T}$，$T$ の終了を $\underline{T}$ と書く（上線は仕事の開始を，下線は仕事の終了を示唆するのに使われている)．ある事象の発生した時刻を $t(\cdot)$ で表すものとし，$t(\underline{T}) - t(\bar{T})$ は，$T$ が必要とする全ての資源が利用可能である限り，正かつ有限であると仮定する．

次に，与えられた $n$ 個のタスクの開始および終了事象の系列を考える．これから展開する理論では，任意に与えられたタスクが行う特定のどのような操

作も仮定しない．すなわち，タスクは解釈なし(uninterpreted)である(タスクは，たとえば，一つのジョブでも，一つのプログラムでも，また一つの命令であってもよい)．したがって本章の結果は，本節で述べる諸定義と矛盾しないどのような(解釈ありの)タスク・システムに対しても適用できる．解釈なしのタスク・システムを研究するもう一つの目的は，得られる結果がそれらのタスクが果す特定の機能に関するものではなく，任意の一群のタスクに対する制御と順序付けに関するものであるという点を強調することである．

タスクを遂行する物理的なシステムには，考察の対象であるシステム内部の態勢(configurations)を表す状態の集合 $S$ が対応する． システムの状態を定義するために， システムはいくつかの**資源**(resource)に分割されており，この分割と資源の明確な定義は考察中の問題に依存してなされると考えれば都合がよい．しかし，一般に資源というものはある機能を遂行するためにタスクによって使われる任意の(必ずしも物理的であるとは限らない)装置であってよい．したがって資源には， 制御装置や入出力装置(テープやディスクの駆動装置， チャネル，カード読取装置など)，プロセッサ，記憶媒体(磁心，ドラム，ディスク，テープなど)の外に，プログラム，手続き，データファイルなどもある．同時に，システムの状態の解釈と定義は，当面問題にしている特定のシステムだけでなく，それに関連した並列計算のモデルの研究における独特の目的をも反映することになる．システムの能力と同様にタスクの機能もまた，タスクの開始と終了の事象に対して定義された許容可能な状態遷移 $s \to s'$ $(s, s' \in S)$ の集合によって明確に示されることになる．

上記の用語において開始の事象は，1)資源の獲得または割当て，2)資源の状態の初期化，3)入力値の読込みを反映する状態遷移に対応している．終了の事象は，1)資源の解放または使用の中止，2)出力値の書出しを反映する状態遷移に対応し， また， 3)値や内部状態が資源に付随している場合には資源(レジスタやページ枠など)の状態を“退避すること” にも対応し得る． これらの準備の下で，タスクの実行が並列に行われるような計算のモデルの定義に移ろう．

$\mathcal{T} = \{T_1, \cdots, T_n\}$をタスクの集合とし，$<$ を $\mathcal{T}$ の上の半順序(先行後続関係， 以下単に先行関係という)とする． 対 $C = (\mathcal{T}, <)$を**タスク・システム**(task system)と呼ぶ．半順序は演算の先行関係を表す． すなわち， $T < T'$ はタスク $T$ がタスク $T'$ の開始よりも前に終了しているべきことを意味する†．

タスク・システムに対して，半順序よりももっと一般的な構造を考えること

† ここで，術語“半順序”は，いくつかの文献に倣って，反射性に関して幾分，誤用している．式 $T < T$ は本書では無意味であり，除外される．

もできるであろう．その一方では，実際に，独立なタスクの集合，すなわち $< = \phi$ である[†]ようなタスク・システム $C = (\mathcal{T}, \phi)$ のみを用いて本章の諸問題を定式化することもできるのである[††]．次章では，スケジューリング問題の取扱いにおいて半順序の一般性を十分に活用するであろう．本章においては本質的ではないけれども，時には，独立なタスクの集合よりも一般的な構造を持っていることが決定的に好都合になることがあるのである．

前述の定義の本質は，グラフを用いてタスク・システムを表現することにより明確になる．タスク・システム $C$ に対応する**先行関係グラフ**(precedence graph，以下単に先行グラフという)は，$\mathcal{T}$ をその節点集合とし，次のような有向枝の集合を持つ．先行グラフにおける $T$ から $T'$ への枝 $(T, T')$ は[†††]，$T < T'$ であり，かつ $T < T'' < T'$ を満たす $T''$ が存在しないときしかもそのときのみ存在する．この定義は先行関係の冗長な指定がグラフに現われないことを保証する．このように，推移的閉包をとると $<$ になるような $\mathcal{T}$ 上の最小の関係として枝の集合を定義することもできる．例を図 2.1-1 に示す．あいまいになる恐れがないとき，しばしばタスク・システムおよび先行グラフをそれぞれ，単に**システム**および**グラフ**と略記する．

グラフ $C$ において節点(タスク)$x_1 \cdots x_k$ を通っている(有向)道$(x_1x_2)(x_2x_3)$

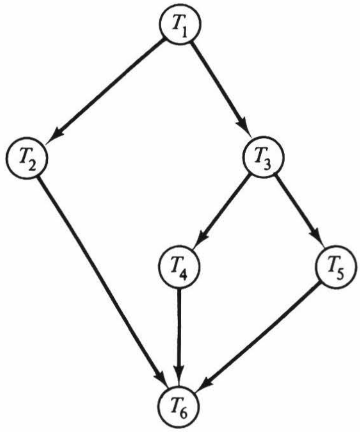



**図 2.1-1**　先行グラフ

† **訳注**　関係 $<$ は $\mathcal{T} \times \mathcal{T}$ の部分集合であることに注意．

†† **訳注**　実際にはそうしていない．

††† **訳注**　タスク名の対 $(T, T')$ は，先行グラフの記述ではその枝を，一般に，タスク・システムの記述では半順序 $<$ の要素を表す．同じ記法であるが，文脈によって読み分けよ．

$\cdots(x_{k-1}x_k)$を考える．この道の**長さ**，すなわち道の上の節点数は $k$ である．また，このとき，$1 \leq i < j \leq k$ であるような $i$ と $j$ に対し，$x_j$ は $x_i$ の**後続者**(または子孫, successor)であり，$x_i$ は $x_j$ の**先行者**(または祖先，predecessor)であるという．$j = i + 1$ であるとき, それぞれ**直接の後続者**(または子, immediate successor)，**直接の先行者**(または親，immediate predecessor)という術語を使う．後続者のないタスクを**終端**(terminal)タスク，先行者のないタスクを**発端**(initial)タスクと呼ぶ．タスク $T$ が $T'$ の後続者でも先行者でもないとき，$T$ と $T'$ は**独立**(independent)であるという．$\lessdot$をタスクについての(時間的な)先行順序であると解釈しているので，$T$ と $T'$ は独立であるときしかもそのときのみ並行的であり得る(そうでなければ $T$ と $T'$ は時間的にある定った順序で生起しなければならない)．非終端タスクは，そこから終端タスクまでの最長の道の長さが $k$ であるとき，$C$ において**レベル**(level)$k$ にあるという．終端タスクのレベルは $1$ であると定義される．

$n$ 個のタスクから成るシステム($n$–**タスク・システム**と呼ぶ)$C = (\mathcal{T}, \lessdot)$の**実行系列**(execution sequence)は，$C$ の先行関係の制約(以下，単に先行制約という)を満たすようなタスクの開始・終了事象の任意の列 $\alpha = a_1a_2 \cdots a_{2n}$ である．正確にいえば,

1. $\mathcal{T}$ に属する各 $T$ に対し, 記号 $\bar{T}$ と $\underline{T}$ が丁度 $1$ 回ずつ $\alpha$ の中に現われる．
2. $a_i = \bar{T}$ かつ $a_j = \underline{T}$ ならば，$i < j$.
3. $T \lessdot T'$，$a_i = \underline{T}$ かつ $a_j = \bar{T}'$ ならば，$i < j$.

図 2.1-1 に対する実行系列の二つの例は,

$$\bar{T}_1\underline{T}_1\bar{T}_2\underline{T}_2\bar{T}_3\underline{T}_3\bar{T}_4\underline{T}_4\bar{T}_5\underline{T}_5\bar{T}_6\underline{T}_6$$

および

$$\bar{T}_1\underline{T}_1\bar{T}_2\bar{T}_3\underline{T}_3\bar{T}_4\bar{T}_5\underline{T}_5\underline{T}_4\underline{T}_2\bar{T}_6\underline{T}_6$$

である．実行系列の任意の語頭(prefix)を**部分**(partial)実行系列という．明らかに，$C$ の実行系列の集合は，事象の系列のうち，$C$ の各タスクが $\lessdot$ によって課せられた先行制約に従いつつ順に遂行することができるようなものの全てから成る集合である．ある $i \leq k$ に対して $a_i = \bar{T}$ でありかつ全ての $j \leq k$ に対して $a_j \neq \underline{T}$ であるとき，タスク $T$ は部分実行系列 $a_1 \cdots a_k$ の後，**活動中**(active)であるという．

問題にしている計算機システムに対する状態空間 $S$ が与えられたとき，実行

系列 $\alpha = a_1a_2 \cdots a_{2n}$ に対応する**状態系列**(state sequence)は，$\sigma = s_0s_1s_2 \cdots s_{2n}$ で表されるが，ここで $s_0$ は与えられた初期状態であり，$s_{i-1} \to s_i$ は事象 $a_i$ に対して定義される状態遷移である．状態と事象の対 $(s, a)$ に対する遷移は，システムの状態が $s$ のときに事象 $a$ が生起可能である場合にのみ定義される．

タスク・システムは唯一の発端タスクと唯一の終端タスクを持つものと仮定すると便利なときがある．この仮定を満たすとき，タスク・システムは**閉じている**(closed)と呼ばれる．図 2.1-1 はは閉じたタスク・システムの例である．$n$–タスク・システムが閉じていないとき，次に示すような“名目だけの(dummy)”タスクである $T_0$ と $T_{n+1}$ を導入することによって閉じたシステム $C'$ を作ることができる．ここで，$T_0$ は $C$ のあらゆる発端タスクの直接の先行者であり，$T_{n+1}$ は $C$ のあらゆる終端タスクの直接の後続者である．$C'$ の実行系列はどれも，$C$ のある実行系列 $\alpha$ に対して $\bar{T}_0\underline{T}_0\alpha\bar{T}_{n+1}\underline{T}_{n+1}$ の形をしている．$T_0$ と $T_{n+1}$ はタスク・システムを閉じさせるだけのために導入されたと仮定すると，システム $C$ と $C'$ は遂行する機能に関して等価である．

$C_1$ と $C_2$ は閉じたタスク・システムであると仮定する．新しい(閉じた)システム $C_1 \cdot C_2$，すなわち，$C_1$ と $C_2$ の**連結**(concatenation)に対するグラフは $C_1$ の終端タスクから $C_2$ の発端タスクへの枝を加えて作られる．$C_1 \cdot C_2$ の実行系列は，$C_1$ のある実行系列 $\alpha_1$ および $C_2$ のある実行系列 $\alpha_2$ に対して，$\alpha = \alpha_1\alpha_2$ であるような任意の列 $\alpha$ である．

$C_1$ と $C_2$ は共通のタスクを持たないシステムであると仮定する．新しいシステムである $C_1$ と $C_2$ の並列結合(parallel combination)$C_1 \| C_2$ は，下記の意味で単に $C_1$ と $C_2$ を一つにまとめたものである．$C_1 \| C_2$ のグラフは，$C_1$ と $C_2$ のグラフを互いに交わらない部分グラフとして持っており，$C_1$ と $C_2$ には共通のタスクがないから，$C_1$ のどのタスクも $C_2$ のあらゆるタスクと独立である．したがって，$C_1 \| C_2$ の実行系列は，$C_1$ と $C_2$ の実行系列を併合することによって作ることができる．より正確に言えば，$c_1c_2 \cdots c_{2(m+n)}$ が $C_1 \| C_2$ の実行系列ならば，次の条件を満たす $C_1$ の実行系列 $a_1a_2 \cdots a_{2n}$ と $C_2$ の実行系列 $b_1b_2 \cdots b_{2n}$ が存在する：

1．$c_1$ は $a_1$ か $b_1$ のどちらかである．
2．$a_1 \cdots a_i$ と $b_1 \cdots b_j$ が $c_1 \cdots c_{i+j}(i + j < 2(m + n))$ の要素であるならば，$c_{i+j+1}$ は $a_{i+1}$ か $b_{j+1}$ のどちらかである．

並列結合の概念はシステム設計においてしばしば現れる．この概念は本章において制御の問題を研究する主な動機を与えている．すなわち，たとえ問題が与

えられたタスク・システムについてそれ用の特別な方法で解けるとしても，一般の解法としては，任意のシステムの並列結合を考慮することが必要である．

多くのオペレーティング・システムには，任意の回数だけ再開始ができるようになっているタスク・システムが内蔵されている．単純かつ典型的な例は，ラインプリンタに出力する行を作り出すループ状のプログラムである．本章で得られる結果のいくつかは先行グラフに閉路がない(acyclic)という性質に依存しているから，そのようなシステムを直接にモデル化するつもりはない．しかし，閉じたタスク・システム $C$ の $k$ 回の反復(iteration)または循環(cycle)は，$k$ 重の連結 $C^k$ としてモデル化できる．このとき $C^k$ の実行系列 $\alpha$ は，$\alpha_i$ を $C$ の $i$ 回目の反復における実行系列としたとき，$\alpha_1\alpha_2 \cdots \alpha_k$ の形をとる．システムが $k > 1$ に対して $C^k$ の形をしているか，または循環的なタスク・システムを少なくとも一つは含むようないくつかの**閉じた**タスク・システムの並列結合であるとき，このシステムは**循環的**(cyclic)であるという．循環的なシステムの実行系列もまた循環的であるという．

以上で，並行プロセスのシステムにおいて生じ得る順序制御の問題を提起し解析する準備ができたことになる．

## 2.2 タスク・システムの決定性

共通の目的に向って協同作業をするタスクの集まりを並列に実行するとき，タスクの実行速度やタスクを実行する順序に変化があっても結果の一意性を保証できるかという問題には，しばしば興味をそそられる．この性質を満たすタスク・システムは**関数的**(functional)である，**速度に依存しない**(speed-independent)，または**決定性**(determinate)であるなどといわれる．タスク・システムは，互いに独立なタスクによって生成される結果がそれらの実行順序に依存するならば，非決定性である．これはよくあることで，たとえば，独立なタスクが共通の記憶位置に読み書きする場合がそうである．非決定性の問題は，前もって独立なタスク間に適当な先行制約を導入しておくことによって簡単に解決する．本節の主な目標は，タスク・システムが決定性であるための必要十分条件を確立することにある．

決定性問題のために，タスク・システムを実行する物理的なシステムを**記憶セル**(memory cell)の順序集合 $M = (M_1, \cdots, M_m)$ で表す．各セル $M_i$ はシステムの**値集合**(value set)と呼ばれる集合 $V$ に属する任意の値を持ち得る．本書の用語は，最近の文献およびオペレーティング・システムの研究における決定性問題の最も重要な応用であると思われるものが動機となっている．しか

し，$M$ の要素は，その状態がタスク・システムによって読ま(使わ)れたり，書か(変えら)れたりする任意の型の資源と見なすことができるということが明らかになろう．より抽象的には，$M$ はタスクに附随した**名前空間**(name space)とも呼ばれる．

システムの状態は，全記憶セル内の値の配置(configuration)で定義される．より正確に言えば，$\alpha = a_1 a_2 \cdots a_{2n}$ と $\sigma = s_0 s_1 \cdots s_{2n}$ をそれぞれ与えられた $n$–タスク・システムの実行系列とそれに対応する状態系列とし，$M_i(k)$ を $k$ 番目の事象の直後の $M_i$ に示される($M_i$ が保持する)値とする．このとき状態 $s_k$ は

$$s_k = [M_1(k), \cdots, M_m(k)]$$

で定義される．したがって，可能な状態全ての集合は，$S = V^m$ と書くことができる．

状態遷移 $s_k \to s_{k+1}$ を定義するためには，タスクの実行が記憶の状態にもたらす効果を先に定式化しておかなければならない．システム $C$ の各タスク $T$ に附随して，一部が重複してもよいような $M$ の二つの部分集合，**定義域**(domain) $D_T$ と**値域**(range) $R_T$ がある．$T$ が開始するとき，$T$ はその定義域セルに蓄えられた値を読み，$T$ が終了するときにはその値域セルに値を書込む．すなわち，$d = |D_T|$, $r = |R_T|$ とすると，各タスク $T$ に附随して写像 $f_T$: $V^d \to V^r$ が存在する．タスク $T$ を実行するということは，写像 $f_T$ を“遂行する”あるいは実現することと見なされるべきである．したがって，与えられた初期状態 $s_0$ と与えられた実行系列 $\alpha$ に対して，対応する状態系列 $\sigma = s_0 s_1 s_2 \cdots s_{2n}$ は次のように定義される．すなわち，$D_T = \{M_{x_1}, \cdots, M_{x_d}\}$，$R_T = \{M_{y_1}, \cdots, M_{y_r}\}$ とすると，

1．$a_{k+1} = \bar{T}$ ならば，$M_i(k+1) = M_i(k) \quad (1 \leq i \leq m)$
2．$a_{k+1} = \underline{T}$ かつ $a_l = \bar{T}\,(l \leq k)$ ならば，

$$[M_{y_1}(k+1), \cdots, M_{y_r}(k+1)] = f_T([M_{x_1}(l), \cdots, M_{x_d}(l)])$$

かつ，全ての $M_i \notin R_T$ に対して，$M_i(k+1) = M_i(k)$.

言い換えると，$a_{k+1}$ が開始事象ならば状態変化は起らず($s_{k+1} = s_k$)，$a_{k+1}$ が $T$ の終了事象ならば $s_{k+1}$ は値域セル $R_T$ においてのみ $s_k$ と異なり得て，その新しい値は $T$ の開始事象の直前の定義域の値に基づき $f_T$ によって定まる．実行系列の結果として生ずる状態系列 $\sigma = s_0 s_1 s_2 \cdots s_{2n}$ は，**履歴配列**(history array)と呼ばれる $m \times (2n+1)$ の配列の形で表現できるが，これは図 2.2-1 に

$C = (\{T_1, T_2\}, \phi)$ $M = (M_1, M_2) = D_{T_1} = D_{T_2} = R_{T_1} = R_{T_2}$

解 釈

| タスク | I | II |
|---|---|---|
| $T_1$ | $(M_1, M_2) \leftarrow (1, M_2)$ | $(M_1, M_2) \leftarrow (M_1 + M_2, M_2)$ |
| $T_2$ | $(M_1, M_2) \leftarrow (2, M_2)$ | $(M_1, M_2) \leftarrow (M_1, M_1 + M_2)$ |

履歴配列

| | | α: | $\overline{T}_1$ | $\underline{T}_1$ | $\overline{T}_2$ | $\underline{T}_2$ | α′: | $\overline{T}_2$ | $\underline{T}_2$ | $\overline{T}_1$ | $\underline{T}_1$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| I | $M_1$ | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 2 | 2 | 1 |
| | $M_2$ | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| II | $M_1$ | 1 | 1 | 3 | 3 | 3 | 1 | 1 | 1 | 1 | 4 |
| | $M_2$ | 2 | 2 | 2 | 2 | 5 | 2 | 2 | 3 | 3 | 3 |

値系列

| | I | | II | |
|---|---|---|---|---|
| | α | α′ | α | α′ |
| $M_1$ | (1, 1, 2) | (1, 2, 1) | (1, 3, 3) | (1, 1, 4) |
| $M_2$ | (2, 2, 2) | (2, 2, 2) | (2, 2, 5) | (2, 3, 3) |

**図 2.2-1** 非決定性タスク・システムの例

示すように行は $M$ のセルに，列は状態に対応している．

以下では，タスク・システムによって個々の記憶セルに書かれる値の系列を取扱う方がより便利である．このために，セル $M_i$ の**値系列**(value sequence) $V(M_i, \alpha)$は，(部分)実行系列 $\alpha$ に現れかつ $M_i \in R_T$ であるようなタスク $T$ の終了によって書かれる値の系列であると定義する．これを正確に表すと，$\epsilon$ は空系列を表すものとし，値系列としてベクトル表記，たとえば，$V(M_i, \alpha)$

$= (v_1, \cdots, v_s)$を採用しよう．$\alpha = a_1a_2 \cdots a_{2n}$ と $\sigma = s_0s_1 \cdots s_{2n}$ を実行系列とそれに対応する状態系列とする．対応する値系列は，次のようになる．

$$V(M_i, \epsilon) = M_i(0)$$

$$V(M_i, a_1 \cdots a_k) = \begin{cases} a_k = \underline{T} \text{ かつ } M_i \in R_T \text{ ならば}(V(M_i, a_1 \cdots a_{k-1}), M_i(k)) \\ \text{そうでなければ } V(M_i, a_1 \cdots a_{k-1}) \end{cases}$$

$M_i$ は $M_j$ と比べて必ずしも同じ数のタスクの値域に入るとは限らないから，$V(M_i, \alpha)$と $V(M_j, \alpha)$は必ずしも同じ長さになるとは限らないことに注意しなければならない．$V(M_i, \alpha) = (v_1, \cdots, v_s)$のとき，**最終値**(final value)$F(M_i, \alpha) = v_s$が定義される．これらの定義に従うと，$\alpha = a_1 \cdots a_k$ ならば $s_k = [F(M_1, \alpha), \cdots, F(M_m, \alpha)]$であることは容易に確かめられる．

半順序$\lessdot$および各タスクの定義域と値域が，タスク・システム $C = (\mathcal{T}, \lessdot)$について知られている全てであると仮定する．このとき，$C$(および $\mathcal{T}$ の各タスク)は**解釈なしである**(uninterpreted)といわれる．タスク・システムに対する**解釈**(interpretation)は，各タスク $T$ に対するタスク関数 $f_T$ の具体的指定(specification)である．一般性を持たせるために，特定のタスク関数 $f_T$ にかかわりなく決定性を保証するタスク・システムの性質を考察しよう．このことから，以下に得られる諸結果は，解釈なしのタスク・システムに対して与えられることになる．

**定義 2.1**

タスク・システム $C$ は，任意に与えられた初期状態 $s_0$ および $C$ の全ての実行系列 $\alpha, \alpha'$ に対して $V(M_i, \alpha) = V(M_i, \alpha')(1 \leq i \leq m)$であるとき，決定性であるという．すなわち，セルの値の系列は，$s_0$ における初期値のみに依存して一意的に定まる．

恐らく，より直観的な決定性の定義は，$F(M_i, \alpha) = F(M_i, \alpha')$であればよいというもっと条件の弱いものであるが，後で述べるように，決定性というタスク・システムの性質は，より強い上述の条件を満たしていなければならない．

**例**：図2.2-1は，記憶セル $M_1$, $M_2$ および解釈 I, II を持つ2-タスク・システムである．履歴配列と値系列が示されている．解釈 I は，タスク $T_1$ と $T_2$ がセル $M_1$ への書込みに競合を生じているから非決定性であり，解釈 II は，あるタスクが読取るセルに他のタスクが書込むから非決定性である．

この例から，あるタスクの値域が他の独立なタスクの定義域または値域と交わるときはいつも，非決定性のシステムになるということは明らかである．直観的には，値域が他の独立なタスクの定義域または値域と交わらないようなタスクから成るシステムは非決定性ではあり得ない．これが次の定義を導き，ひいては決定性システムであるための必要十分条件を導くことになる．

**定義 2.2**

タスク $T$ および $T'$ は，次のいずれかの条件を満たすとき，**非干渉である**(non-interfering)という．

1. $T$ が $T'$ の後続者であるか先行者である．
2. $R_T \cap R_{T'} = R_T \cap D_{T'} = D_T \cap R_{T'} = \phi$.

また，$\mathcal{T} = \{T_1, \cdots, T_n\}$ は，全ての $i$ と $j(i \neq j)$に対して $T_i$ と $T_j$ が非干渉であるとき，互いに非干渉なタスクから成っているという．

次の補題は，互いに非干渉なタスクから成っているシステムが決定性であることを保証する定理2.1の証明に用いる帰納法の第一段を与える．

**補題 2.1**

$C$ を互いに非干渉なタスクから成る $n$–タスク・システムとし，$T$ を $C$ の一つの終端タスクとする． $\alpha = \beta_1 \bar{T} \beta_2 \underline{T} \beta_3$ が $C$ の実行系列ならば， $\alpha' = \beta_1 \beta_2 \beta_3 \bar{T} \underline{T}$ も $C$の実行系列であり，全ての $i(1 \leq i \leq m)$に対して $V(M_i, \alpha) = V(M_i, \alpha')$である．

**証明** $C$ には $T$ の後続者がないから$\alpha'$ は $C$ の先行制約を満たしており，それゆえ $\alpha'$ も一つの実行系列でなければならない．$T$ は $R_T$ のセルにのみ書込みを行い，かつ $\beta_2$ 内で開始される任意の $T'$ に対して $R_T \cap D_{T'} = \phi$ であるから，そのような各 $T'$ の $\alpha'$ における定義域の値は $\alpha$ における場合と同じである．したがって，$M_i \notin R_T$ に対して $V(M_i, \alpha) = V(M_i, \alpha')$.

さて，任意の $M_j \in D_T$ に対し，($R_{T'} \cap D_T = \phi$ であるので)$\beta_2 \beta_3$ 内には $D_T$ に書込む $T'$ は存在しないから， $V(M_j, \beta_1) = V(M_j, \beta_1 \beta_2 \beta_3)$. したがって，全ての $M_j \in D_T$ に対し，$F(M_j, \beta_1) = F(M_j, \beta_1 \beta_2 \beta_3)$であり， $T$ は $\alpha'$ においても $\alpha$ のときと同じ値を各 $M_i \in R_T$ に書込む．$\alpha$ において $T$ によって $M_i \in R_T$ に書込まれる値を $v$ で表すと，次のようになる．

$V(M_i, \alpha) = V(M_i, \beta_1 \bar{T} \beta_2 \underline{T})$ ($\beta_3$ には $R_T$ に書込む $T'$ がない)

$$= (V(M_i,\ \beta_1\bar{T}\beta_2),\ v) \qquad (T \text{ は } M_i \text{ に } v \text{ を書込む})$$
$$= V(M_i,\ \beta_1),\ v) \qquad (\beta_2 \text{ には } R_T \text{ に書込む } T' \text{ がない})$$
$$= (V(M_i,\ \beta_1\beta_2\beta_3),\ v) \qquad (\beta_2\beta_3 \text{ には } R_T \text{ に書込む } T' \text{ がない})$$
$$= V(M_i,\ \beta_1\beta_2\beta_3\bar{T}\underline{T}) \qquad (T \text{ は } M_i \text{ に } v \text{ を書込む})$$
$$= V(M_i,\ \alpha')$$

∎

素直な帰納法の議論により互いに非干渉なタスクであるということは決定性であるための十分条件であることが立証される．

**定理 2.1**

互いに非干渉なタスクから成るタスク・システムは決定性である．

**証明**　システム内のタスクの数に関する帰納法で議論を進めよう．定理の主張は1–タスク・システムでは明らかに真である．タスク数が$m$未満のシステムに対してこの主張が成り立つと仮定し，$C = (\mathcal{T},\ <)$を $n$–タスク・システムであるとする．

$C$ は，実行系列が一つしかなければ明らかに決定性である．それゆえ，$\alpha_1$ と $\alpha_2$ が $C$ の異なる実行系列であると仮定しよう．$T$ を $C$ の一つの終端タスクとし，補題2.1 にならって $\alpha'_1$ と $\alpha'_2$ を作ると次式を得る．

$$\alpha'_1 = \alpha''_1\bar{T}\underline{T};\ V(M_i,\ \alpha_1) = V(M_i,\ \alpha'_1) \quad (1 \le i \le m)$$
$$\alpha'_2 = \alpha''_2\bar{T}\underline{T};\ V(M_i,\ \alpha_2) = V(M_i,\ \alpha'_2) \quad (1 \le i \le m)$$

しかし，$T$ が関与する全ての先行制約($T <$・および・$< T$)を $<$ から除いて得られる先行関係を $<'$ とすると，$\alpha''_1$ と $\alpha''_2$ はいずれも $(n-1)$–タスク・システム $C' = (\mathcal{T} - \{T\},\ <')$の実行系列となるから，帰納法の仮定により，$V(M_i,\ \alpha''_1) = V(M_i,\ \alpha''_2)\ (1 \le i \le m)$．したがって，$T$ で用いられる定義域の値は $\alpha'_1$ においても $\alpha'_2$ においても同じである．すなわち，$M_j \in D_T$ に対して，$F(M_j,\ \alpha''_1) = F(M_j,\ \alpha''_2)$，このことから，$\alpha'_1$ と $\alpha'_2$ のいずれの場合にも，$T$ は $M_i \in R_T$ に同じ値を書くはずである．したがって，$M_i \in R_T$ に対しては，

$$V(M_i,\ \alpha_1) = V(M_i,\ \alpha'_1) \qquad (\text{補題}\,2.1)$$
$$= (V(M_i,\ \alpha''_1),\ v) \qquad (T \text{ は } M_i \text{ に } v \text{ を書込む})$$
$$= (V(M_i,\ \alpha''_2),\ v) \qquad (\text{帰納法の仮定})$$
$$= V(M_i,\ \alpha'_2) \qquad (T \text{ は } M_i \text{ に } v \text{ を書込む})$$

$$= V(M_i,\ \alpha_2) \qquad (\text{補題 } 2.1)$$

また，$M_i \notin R_T$ に対しては，

$$\begin{aligned} V(M_i,\ \alpha_1) &= V(M_i,\ \alpha'_1) && (\text{補題 } 2.1) \\ &= V(M_i,\ \alpha''_1) && (M_i \notin R_T) \\ &= V(M_i,\ \alpha''_2) && (\text{帰納法の仮定}) \\ &= V(M_i,\ \alpha'_2) && (M_i \notin R_T) \\ &= V(M_i,\ \alpha_2) && (\text{補題 } 2.1) \end{aligned}$$

以上によって，$C$ は決定性であることが証明された．■

非干渉性は，タスク・システムが次の2条件を満たしているとき，決定性であるための必要条件でもある．

1．どのタスクも空の値域を持たない．なぜならば，もしそうでないとすると，$R_T = \phi$ であるようなタスク $T$ は干渉され得るが，このことはセルの内容を観察することによって検出する方法はない．

2．決定性は，一つの(解釈なしの)システムの全ての可能な解釈に対して成立しなければならない．

**定理 2.2**

タスク・システム $C$ において，どのタスク $T$ に対しても，$f_T$ の具体的指定は与えられていないが，$R_T(\neq\phi)$ と $D_T$ は与えられているものとする．このとき，タスクの全ての解釈に対して $C$ が決定性であるならば，タスクは互いに非干渉である．

**証明** 全ての解釈に対して $C$ が決定性であるのに互いに干渉するタスクが存在することを仮定して矛盾を導こう．$T$ と $T'$ が干渉し合うとするとこの両者は互いに独立であって，$C$ には次のような実行系列が存在する．

$$\alpha = \beta_1 \bar{T}\, \underline{T}\, \bar{T}'\, \underline{T}'\, \beta_2$$
$$\alpha' = \beta_1 \bar{T}'\, \underline{T}'\, \bar{T}\, \underline{T}\, \beta_2$$

まず，$R_T \cap R_{T'} \neq \phi$ と仮定し，$M_i \in (R_T \cap R_{T'})$ であると仮定する．この場合，$T$ が $M_i$ に $u$ を書込み，$T'$ が $v(\neq u)$ を書込むような $f_T$ と $f_{T'}$ を選ぶことができる(たとえば図 2.2-1 を見よ)．しかし，

$$V(M_i,\ \beta_1\bar{T}\underline{T}\bar{T}'\underline{T}') = (V(M_i,\ \beta_1),\ u,\ v)$$
$$V(M_i,\ \beta_1\bar{T}'\underline{T}'\bar{T}\underline{T}) = (V(M_i,\ \beta_1),\ v,\ u)$$

となり，$C$ が決定性であることと矛盾する．

次に，ある $M_j$ が存在して $M_j \in (R_{T'} \cap D_T)$ であると仮定する．$M_i \in R_T$ ($R_T \neq \phi$ ゆえそのような $M_i$ は必ず存在する)とする．このとき，$F(M_j,\ \beta_1) \neq F(M_j,\ \beta_1\bar{T}'\underline{T}')$ であるような，すなわち $\alpha$ と $\alpha'$ で $T$ が異なる値を $M_j$ から読取るような $f_{T'}$ を選ぶことができる．その結果，$T$ が $\alpha$ では $u$ を，$\alpha'$ では $v(\neq u)$ を $M_i$ に書込むように $f_T$ を選ぶことができる(たとえば図**2.2-1**を見よ)．しかし，そのときには，

$$\begin{aligned} V(M_i,\ \beta_1\bar{T}\underline{T}\bar{T}'\underline{T}') &= V(M_i,\ \beta_1\bar{T}\underline{T}) \qquad (R_T \cap R_{T'} = \phi) \\ &= (V(M_i,\ \beta_1),\ u) \\ V(M_i,\ \beta_1\bar{T}'\underline{T}'\bar{T}\underline{T}) &= (V(M_i,\ \beta_1\bar{T}'\underline{T}'),\ v) \\ &= (V(M_i,\ \beta_1),\ v) \qquad (R_T \cap R_{T'} = \phi) \end{aligned}$$

となり $u \neq v$ ゆえ，$C$ が決定性であるという仮定と矛盾する．(もし $R_T = \phi$ ならば，$M_j \in (R_{T'} \cap D_T)$ に対してこの矛盾に到達させ得ないことに注意しよう．)

さらに，対称性から，$M_j \in (R_T \cap D_{T'})$ を仮定した場合にも矛盾が生ずる．以上により，$T$ と $T'$ が干渉し得る全ての場合について矛盾が生ずることになり背理法による証明が完結する．■

定理2.1と定理2.2が役に立つ一つの応用は，決定性であるべきであるという必要性のみから先行制約が定まるようなタスク・システムの並列性，すなわち最大並列性 [26] に関するものである．この応用を説明するために，まず，最大並列性の概念とタスク・システム間の等価性の概念を形式化しよう．

定義2.1から，決定性のタスク・システムおよび与えられた初期状態に対し，各記憶セル $M_i$ に対する値系列がそれぞれ正確に一つずつ対応する．タスクの集合が同じである二つのタスク・システムは，いずれも決定性であり，かつ，同じ初期状態に対して同じ値系列を生成するとき，**等価である**(equivalent)という．タスク・システム $C$ とそのグラフ $G$ は，$C$ が決定性であり，かつ，$G$ からどの枝 $(T,\ T')$ を取り除いても $T$ と $T'$ が干渉することになるとき，**最大並列**(maximally parallel)であるという．それゆえ，$(T,\ T')$ が最大並列なグラフの枝であるならば，次式が成り立つ．

$$(2.2.1) \qquad (R_T \cap R_{T'}) \cup (R_T \cap D_{T'}) \cup (R_{T'} \cap D_T) \neq \phi$$

決定性のタスク・システム $C$ が与えられたとき，それと等価な最大並列のタスク・システム $C'$ を構成する問題は興味深い．

**定理 2.3**

与えられた決定性のタスク・システム $C = (\mathcal{T}, \lessdot)$ から，新しいタスク・システム $C' = (\mathcal{T}, \lessdot')$ を構成する．ただし，$\lessdot'$ は次式で与えられる関係 $X$ の推移的閉包である．

$$X = \{(T, T') \in \lessdot \mid (R_T \cap R_{T'}) \cup (R_T \cap D_{T'}) \cup (R_{T'} \cap D_T) \neq \phi\}$$

このとき，$C'$ は $C$ と等価であり，しかも一意に定まる最大並列なシステムである．

**証明** $X$ に，したがってまた $\lessdot'$ にも対応する先行グラフを $G'$ で表す（図 2.2-2 の例は，一般に $G' \subset X \subset \lessdot'$ であることを説明している）．$G'$ から枝 $(T, T')$ を取除いたと仮定しよう．このとき，（冗長な枝はないという先行グラフの定義より）$T$ から $T'$ への道は他には存在しないこと，および，$X(\supset G')$ の定義より $T, T'$ ともに式(2.2.1)を満たすことから，$T$ と $T'$ は干渉し合うようになることが分る．したがって，もし $C'$ が決定性であることを示すことができれば，$G'$ は最大並列であるといえる．

$C$ が決定性であるから，式(2.2.1)を満たすどのようなタスクの対 $T$ と $T'$ に対しても，$T \lessdot T'$ であるか $T' \lessdot T$ であるかのいずれかである．また，$X$ の定義から，$T \lessdot' T'$ か $T' \lessdot' T$ のいずれかが成り立つ．したがって，$C'$ の全てのタスクは互いに非干渉であり，定理 2.1 から $C'$ も決定性である．

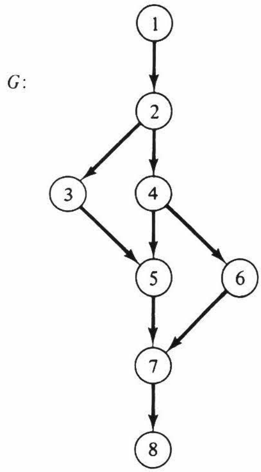
$C'$ の先行制約はことごとく $C$ にもある．したがって，$C$ の実行系列はいずれも $C'$ によって生成される．$C$ では生成できないような $C'$ の実行系列を $\alpha'$ とし，$C$ と $C'$ に共通の実行系列を $\alpha$ とする．$C'$ が決定性であることから，与えられた初期状態に対して，$V(M_i, \alpha') = V(M_i, \alpha)$ $(1 \leq i \leq m)$ が成り立つ．それゆえ，$C'$ は $C$ と同じセルの値系列を持ち，$C$ と $C'$ は等価であると結論できる．

最後に，$T \lessdot' T'$ であるときかつそのときのみ，タスクの連鎖 $T = T_{i_1} \lessdot T_{i_2} \lessdot \cdots \lessdot T_{i_k} = T'$ $(k > 1)$ が存在し，そこに現れるあらゆるタスクの対 $T_{i_j}$ と $T_{i_{j+1}}$ $(1 \leq j < k)$ は式(2.2.1)を満たす．これが $C$ に等価な任意の最大並列なタスク・システムについていつも成り立たねばならないことは容易に確かめるこ

$$M = \{M_1, M_2, M_3, M_4, M_5\}$$

（タスクはその番号で表わされている）

| セル | 定義域内になっているタスク | 値域内になっているタスク |
|---|---|---|
| $M_1$ | 1, 2, 7, 8 | 3 |
| $M_2$ | 1, 7 | 5 |
| $M_3$ | 3, 4, 8 | 1 |
| $M_4$ | 3, 4, 5, 7 | 2, 7 |
| $M_5$ | 6 | 4, 6, 8 |




$X$:

(1, 3)　(1, 4)　(1, 5)　(1, 8)
(2, 3)　(2, 4)　(2, 5)　(2, 7)
(3, 7)　(3, 8)
(4, 6)　(4, 7)　(4, 8)
(5, 7)
(6, 8)

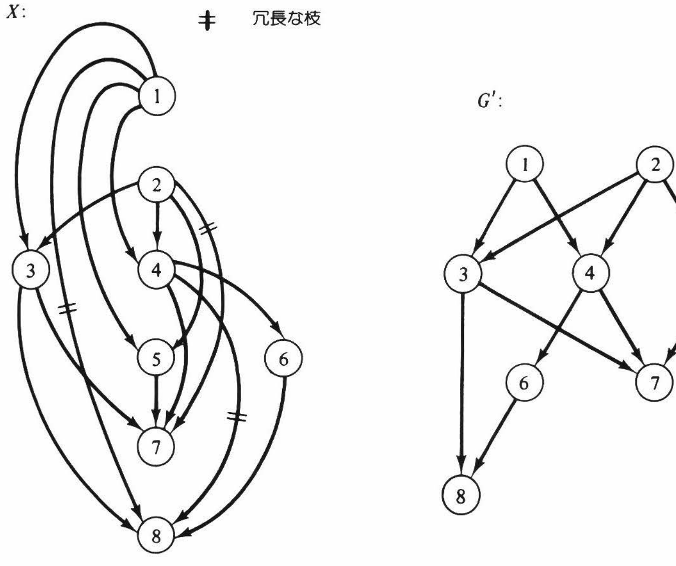



**図 2.2-2**　最大並列なグラフの算出

とができる．このことから，$C'$ は一意である．■

図2.2-2は，$G'$ の構成法を示している(問題2.10参照)．関係 $X$ を算出した後，対応するグラフが描かれる．このグラフから冗長な枝を取除くと $G'$ が得られる．[枝$(T, T')$は，2本以上の枝を含む $T$ から $T'$ への別の道が存在するとき，冗長である.] $X$ の算出は，$T_i < T_j$ ならば $i < j$ であるといったタスクの番号の付け方でいくらか簡単になることに注意しよう(問題2.11参照)．そうすると，$i = 1, 2, \cdots n$，かつ $j = i + 1, \cdots n$ であるようなタスクの対$(T_i, T_j)$のみを考えて，式(2.2.1)が成り立つときにはいつもこの対を含めるという方法で $X$ を構成することができる．

決定性に関する諸結果は沢山の他の文献にも見られるが，オペレーティング・システムの環境に関連した非形式的な論述は Bernstein [1] によって与えられた．Bernstein は，プログラム(たとえば Algol で表現された)を並列に実行可能なタスクに分解する一般の問題が帰納的に可解ではない(recursively unsolvable)ことを証明した．その証明の接近法は決定問題に対する典型的な方法によっているが，本書が紹介する範囲外である．Baer [2] は，並列性についての種々の局面——たとえば，並列に実行することができるプログラムのブロックの検出，並列性に関する他のモデル，ハードウェアでの並列性の実現，並列性を表現するためのプログラム言語面での方法など——を含む広範な概説を与えている(問題2.12，2.14参照)．

## 2.3 閉　鎖†

### 2.3.1 はじめに

タスク・システム $C = C_1 \| C_2 \| \cdots \| C_n$ は，次のようなときに閉鎖(deadlock)状態に陥る．すなわち，$\{C_i | 1 \leq i \leq n\}$ のある部分集合 $S$ に属しているシステムの各々が同時に活動中であり，しかも $S$ に属する各システムが進行するために獲得しなければならない横取り不可の資源を $S$ 内の他のシステムが保持していることが原因になって，$C$ の進行が無定限(indefinitely)に阻止されるときである．本章で横取り不可の資源というのは，それを保持しているタスク・システムによってのみ解放され得る資源のことである．

閉鎖の問題を研究するために，$C$ が乗る計算機システムはそれぞれ $w_1, \cdots, w_m$ 個の単位(装置)を持つ $m$ 種の**資源の型**(resource type)を有しているもの

† この問題に関する概説が文献[3]に与えられている．

とする．ベクトル $\boldsymbol{W} = (w_1, \cdots, w_m)$は，システムの容量(capacity)を表す．与えられた型の資源の個々の単位を識別することは不可能であるものとする．資源の型の例には，プロセッサ，記憶のページ，テープ装置，ディスクのトラックなどがあるが，ハードウェア装置それ自体ではないような資源，たとえば，データ・ファイル・メッセージ緩衝記憶やシステムが提供する表なども含まれよう．資源が横取り不可であるということは，単に与えられた資源に横取り権が及ぶようにはシステムが設計されていないことによる場合もあるし，本質的な制約に起因する場合もある．後者の例として，更新中のデータ・ファイルや，機械的な位置決めを伴う入出力装置などがある．

どのような条件の下で閉鎖が生ずるのかを説明するために，簡単な例を考える[4]．二つのシステム $C_1$, $C_2$ に対して型 $R_1$, $R_2$ の資源がそれぞれ1単位ずつあるとする．各システムについて，その開始以来完了した命令数でその進行の度合を示すものとすると，そのような値の対は図2.3-1に示すように，2次元の**進行空間**(progress space)上の点を定義する．$C_1$ と $C_2$ を結合した進行は，この空間内の離散的な点の系列$(x_1, y_1)(x_2, y_2)\cdots(x_k, y_k)\cdots$で表される．明らかに，$x_k \leq x_{k+1}$ かつ $y_k \leq y_{k+1}$ である(すなわち，進行は可逆ではない)．(0, 0)から始まる進行点のそのような系列は，$C_1$ と $C_2$ の**結合進行路**(joint progress path)と呼ばれる．

あらゆる結合進行路が実行可能であるとは限らない．図2.3-1に示されるように，$R_1$ は1単位しか使えないから，$R_1$ を同時に使うことに対応する進行点は許されない．いかなる進行路も，それが正当† であるためには実行不可能な

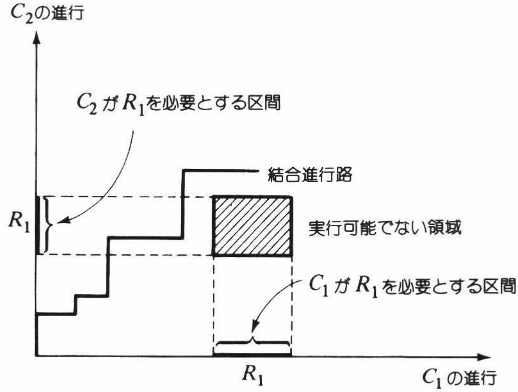



**図2.3-1　進 行 空 間**

† **訳注**　この術語の厳密な定義は後でなされる．

領域($R_1$ 領域という)を避けて通らなければならない.

さて, **図 2.3-2** を考えよう. この図は $R_1$ 領域と $R_2$ 領域の合併によって生ずるより大きな実行不可能領域を示している. もし, 結合進行路が一度領域 $U$ ——"安全でない領域(unsafe region)"——に入ろうものなら, 進行の非可逆性から, 究極的には閉鎖が発生することは避けられない. 明確にいえば, どちらのタスクの進行も $U$ の右上の角で永久に停止しなければならない. 安全でない領域が存在するためには三つの条件が必要である.

1. **相互排除**(Mutual exclusion). 各タスク・システムは, それが使用する資源を排他的に自己の制御下に置くことを主張する.
2. **横取り権なし**(Nonpreemption). タスク・システムは保持している資源の使用が終るまでそれを解放しない.
3. **資源待ち**(Resource waiting). 各タスク・システムは, 他のタスク・システムが資源を解放するのを待っている間, (既に獲得した)資源を保持し続ける.

閉鎖それ自身は, 解けない**循環待ち**(circular wait)の状態として定義されるけれども, この待ちが解けないことは相互排除と横取り権なしの条件で保証されている. **進行**(progress), **閉鎖**(deadlock)および**安全性**(safeness)の概念を形式化した後, 閉鎖の扱いに関する次の三つの接近法を考えよう.

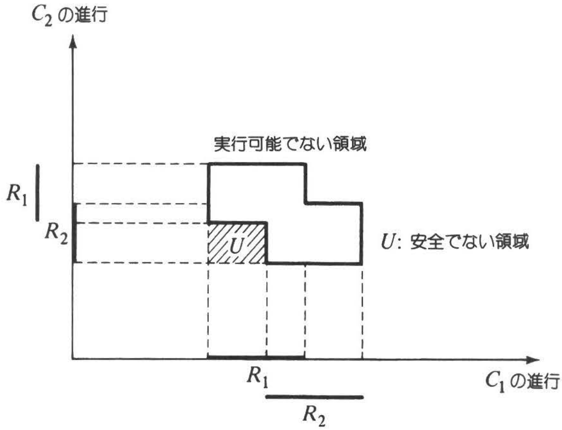



**図 2.3-2** 安全でない領域がある進行空間

1．**予防**(Prevention)．システム設計を規制するか，資源の使い方を適当に制限することによって，上記の三つの必要条件のうちの一つかそれ以上が除外されるようにする．

2．**検出と回復**(Detection and recovery)．オペレーティング・システムが全てのタスク・システムの結合進行を監視していて，もし閉鎖の発生が察知されたならば，最小の費用でこの事態を矯正し回復するための特別な動作を行うようにする．

3．**回避**(Avoidance)．資源の使い方について予め分っている情報に基づいて，オペレーティング・システムは，結合進行路の次の一歩が安全でない領域に入らないように，資源の割当てを制御する．

閉鎖の問題を形式化するに当って，タスクの**連鎖**(chain)のいくつかから成るタスク・システムを考える [5]．連鎖のシステム $C_1 \parallel C_2 \parallel \cdots \parallel C_n$ 中の $i$ 番目の連鎖を $C_i = T_i(1)\,T_i(2)\cdots T_i(p_i)\ (p_i \geq 1)$ と表そう．閉鎖の性質に関連してタスク・システムは連鎖の並列結合であるとする単純化は，本質的には一般性を全く損なうことのない大変都合のよいものである．重要な仮定は，連鎖の資源の使い方は一つのタスクの実行中は不変であるけれども，実行があるタスクから他のタスクへ移るときには資源の使い方も変化するということである．実質的にこの性質が，連鎖によってモデル化されているプログラムまたはジョブのタスクへの分解を定義しているのである．

各タスクは，その連鎖によって既に保持されている資源に加えて，実行開始前に行う資源単位の要求と，保持している資源のうち連鎖内の次のタスクへ引渡さない資源単位に対して実行終了後に行う解放とによって特徴づけられる．したがって，タスク $T_i(l)$ には**要求ベクトル**(request vector) $\mathbf{q}_i(l) = (q_{i1}(l), \cdots, q_{im}(l))$ と**解放ベクトル**(release vector) $\mathbf{r}_i(l) = (r_{i1}(l), \cdots, r_{im}(l))$ が対応づけられる．ここで，$q_{ij}(l)$ は，$T_i(l)$ の開始に必要な $R_j$ の追加単位数を，$r_{ij}(l)$ は $T_i(l)$ の終了時に解放される $R_j$ の単位数を表す．また，$m$-組の $\mathbf{0}$ を $\mathbf{0}$ で表すことにする．一般には，$\mathbf{q}_i(l) = \mathbf{0}$ あるいは $\mathbf{r}_i(l) = \mathbf{0}$ であってもよい．連鎖 $C_i$ の資源の使い方が $T_i(l)$ と $T_i(l+1)$ で変るという仮定は次式と等価である．

$$\mathbf{r}_i(l) \neq \mathbf{q}_i(l+1) \quad (1 \leq l < p_i)$$

まず，一本の連鎖 $C_i$ が実行されるという仮定の下に，状態と状態遷移を定義しよう．連鎖 $C_i$ は唯一の実行系列 $\alpha = \bar{T}_i(1)\,\underline{T}_i(1)\cdots \underline{T}_i(p_i) = a_1 \cdots a_k \cdots a_{2p_i}$ を持つので，これに対応する状態系列 $\sigma = s_0 s_1 \cdots s_k \cdots s_{2p_i}$ を明確にした

い．そこで，これらの状態を，$\alpha$ の各事象の後で連鎖に割当てられている資源の集合と，連鎖が要求している資源の集合がそれぞれ明示されるように定義しよう．閉鎖の研究はタスクが遂行する機能に関わりがないから，資源の内部状態は問題に関与しない．それゆえ，一対のベクトル $\mathbf{P}_i(k)=(P_{i1}(k),\cdots,P_{im}(k))$ と $\mathbf{Q}_i(k)=(Q_{i1}(k),\cdots,Q_{im}(k))$ によって状態 $s_k$ が完全に指定される．ここで，$P_{ij}(k)$ と $Q_{ij}(k)$ はそれぞれ，$\alpha$ の事象 $a_k$ の後，連鎖 $C_i$ が保持している $R_j$ の単位数と要求している $R_j$ の単位数を表す．ベクトル $\mathbf{P}_i(k)$ と $\mathbf{Q}_i(k)$ は，$\mathbf{q}_i(l)$ と $\mathbf{r}_i(l)$ を用いて次のように表される．

$$
\begin{aligned}
&\mathbf{Q}_i(0)=\mathbf{q}_i(1) \quad , \ \mathbf{P}_i(0)=0\\
&a_k=\bar{T}_i(l) \text{ ならば，}\\
(2.3.1) \qquad &\quad \mathbf{Q}_i(k)=0 \quad , \ \mathbf{P}_i(k)=\mathbf{P}_i(k-1)+\mathbf{Q}_i(k-1)\\
&a_k=\underline{T}_i(l) \text{ ならば，}\\
&\quad \mathbf{Q}_i(k)=\mathbf{q}_i(l+1),\ \mathbf{P}_i(k)=\mathbf{P}_i(k-1)-\mathbf{r}_i(l)
\end{aligned}
$$

資源の最終的な割当て状態を表す $\mathbf{P}_i(2p_i)$ は次の条件を満たすものとする．

$$
(2.3.2) \qquad \mathbf{P}_i(2p_i)=\sum_{l=1}^{p_i}\mathbf{q}_i(l)-\mathbf{r}_i(l)=0
$$

（すなわち，連鎖は究極的には，それが要求したあらゆる資源を解放する．）式(2.3.1)によれば，タスクが要求した追加分の資源は，同時に与えられなければならない．また終了事象 $\underline{T}_i(l)$ には，$T_i(l)$ による資源の解放とその直後に $T_i(l+1)$ によってなされる資源の要求とが結び付けられている．さらに，全ての資源が利用可能であるように初期状態を選んだのは，その方が便利だからというだけに過ぎない．以下の諸定義と諸結果は，より一般的な初期状態を考慮に入れたものに容易に拡張される．図 2.3-3 は，三つのタスクから成る連鎖 $C_1$ と，それらのタスクの要求/解放ベクトルを示している．$C_1$ の実行系列に対応する状態系列も同図に示されている．

さて，システム $C=C_1 \| C_2 \| \cdots \| C_n$ が実行されるものとして，状態および状態遷移を定義しよう．上記の議論を一般化すると，システムの各状態 $s_k$ は，次に示す行列の対 $[\mathsf{P}(k),\mathsf{Q}(k)]$ によって表される．

$$
\mathsf{P}(k)=\begin{bmatrix}\mathbf{P}_1(k)\\ \cdot\\ \cdot\\ \cdot\\ \mathbf{P}_n(k)\end{bmatrix} \qquad \mathsf{Q}(k)=\begin{bmatrix}\mathbf{Q}_1(k)\\ \cdot\\ \cdot\\ \cdot\\ \mathbf{Q}_n(k)\end{bmatrix}
$$



| | $C_1$ | $\mathrm{q}_1$ | $\mathrm{r}_1$ |
|---|---|---|---|
| $T_1(1)$ | | (0, 1, 2, 0) | (0, 0, 2, 0) |
| $T_1(2)$ | | (1, 0, 0, 1) | (0, 0, 0, 1) |
| $T_1(3)$ | | (0, 0, 2, 0) | (1, 1, 2, 0) |

システム：$\mathbf{R} = (R_1, R_2, R_3, R_4)$,　$\mathbf{w} = (3, 3, 3, 3)$

| | | 実行系列 | $\overline{T}_1(1)$ | $\underline{T}_1(1)$ | $\overline{T}_1(2)$ | $\underline{T}_1(2)$ | $\overline{T}_1(3)$ | $\underline{T}_1(3)$ |
|---|---|---|---|---|---|---|---|---|
| 状態 | $\mathrm{P}_1(k)$: | (0000) | (0120) | (0100) | (1101) | (1100) | (1120) | (0000) |
| | $\mathrm{Q}_1(k)$: | (0120) | (0000) | (1001) | (0000) | (0020) | (0000) | (0000) |

**図 2.3-3**　連鎖と実行系列の例

$\alpha = a_1 \cdots a_k \cdots$ がシステム $C$ の実行系列ならば，対応する状態系列 $s_0 s_1 \cdots s_k \cdots$ は式(2.3.1)を行列 $\mathrm{P}(k)$ と $\mathrm{Q}(k)$ の適当な行に適用することによって定義される．これらの行列において，$\mathbf{Q}_i(k) \neq \mathbf{0}$ の持つ意味は，連鎖 $C_i$ が一つかそれ以上の資源の割当てを待っているということである．また，$\mathbf{P}_i(k) \neq \mathbf{0}$ かつ $\mathbf{Q}_i(k) = \mathbf{0}$ は，$C_i$ が実行中であるかまたは実行を続ける準備が整っていることを意味する．$\mathbf{P}_i(k) + \mathbf{Q}_i(k) \neq \mathbf{0}$ であるならば，連鎖 $C_i$ は**活動中**であるという．二つの連鎖を持つタスク・システムの例を図 2.3-4 に示す．

我々のモデルでは，考慮中の連鎖の集合にタスクの連鎖を追加することも削除することもできないという意味で，十分な一般性に欠けている．後で分るように，行列 P と Q に行を追加したり削除したりすることによって新しい連鎖の参入や退去を考慮に入れることは容易である．特に，新しい連鎖 $C_{n+1}$ が活動中の連鎖の集合に参入するとき，行列 P には行 $\mathbf{P}_{n+1} = \mathbf{0}$ が，行列 Q には行 $\mathbf{Q}_{n+1} = \mathbf{q}_{n+1}(1)$ が追加される．これは本節の議論の自然な拡張であり，定義や結果に本質的にはどのような影響も与えないから，これ以上の考察は行わない．

開始の遷移(あるタスクの開始に対応する遷移)は，どの型 $R_j$ についてもそのタスクが要求している資源量がその時点で利用可能な $R_j$ の量を越えていないとき，(次に定義する $\mathbf{v}(k)$ に関して)**許容可能である**(allowable)という．**利用可能資源ベクトル**(available resource vector) $\mathbf{v}(k) = (v_1(k), \cdots, v_m(k))$ を

| | $C_1$ | | | $C_2$ | |
|---|---|---|---|---|---|
| | $\mathbf{q}_1$ | $\mathbf{r}_1$ | | $\mathbf{q}_2$ | $\mathbf{r}_2$ |
| $T_1(1)$ | (0, 1, 2, 0) | (0, 0, 2, 0) | $T_2(1)$ | (2, 0, 2, 2) | (0, 0, 0, 2) |
| $T_1(2)$ | (1, 0, 0, 1) | (0, 0, 0, 1) | $T_2(2)$ | (1, 0, 1, 0) | (3, 0, 3, 0) |
| $T_1(3)$ | (0, 0, 2, 0) | (1, 1, 2, 0) | | | |

システム：　$\mathbf{R}=(R_1, R_2, R_3, R_4)$,　$\mathbf{w}=(3, 3, 3, 3)$

| 実行系列 | $\alpha$ | $\bar{T}_1(1)$ | $\underline{T}_1(1)$ | $\bar{T}_2(1)$ | $\underline{T}_2(1)$ | $\bar{T}_2(2)$ | $\underline{T}_2(2)$ |
|---|---|---|---|---|---|---|---|
| P(*k*): | 0000<br>0000 | 0120<br>0000 | 0100<br>0000 | 0100<br>2022 | 0100<br>2020 | 0100<br>3030 | 0100<br>0000 |
| Q(*k*): | 0120<br>2022 | 0000<br>2022 | 1001<br>2022 | 1001<br>0000 | 1001<br>1010 | 1001<br>0000 | 1001<br>0000 |

| $\bar{T}_1(2)$ | $\underline{T}_1(2)$ | $\bar{T}_1(3)$ | $\underline{T}_1(3)$ | |
|---|---|---|---|---|
| 1101<br>0000 | 1100<br>0000 | 1120<br>0000 | 0000<br>0000 | |
| | | | | 完　了 |
| 0000<br>0000 | 0020<br>0000 | 0000<br>0000 | 0000<br>0000 | |

**図 2.3-4**　$C=C_1||C_2$ の例

次のように定義する．

$$v_j(k)=w_j-\sum_{i=1}^{n} P_{ij}(k) \quad (1\le j\le m)$$

$T_i(l)$の開始が連鎖 $C_i$ で次に起ろうとしている事象であると仮定すると，事象 $a_{k+1}=\bar{T}_i(l)$に対応する状態遷移は，$\mathbf{Q}_i(k)\le\mathbf{v}(k)$† ならば許容可能である．実行系列は，その開始の遷移の全てが許容可能であるとき**正当である**(valid)という．正当な実行系列が常に存在することは次のようにして示される．どの連鎖もシステムの容量を超える資源の要求を決してしないことを思い出そう．任意の順序で(たとえば，$C_1$ が終れば $C_2$, $C_2$ が終れば $C_3$ といった順に)連鎖

† $\mathbf{x}=(x_1, \cdots, x_m)$, $\mathbf{y}=(y_1, \cdots, y_m)$ であるとき，$\mathbf{x}\le\mathbf{y}$ は各 $i$ に対して $x_i\le y_i$ であることを意味する．また，$\mathbf{x}<\mathbf{y}$ は，$\mathbf{x}\le\mathbf{y}$，かつ，少なくとも一つの $i$ に対して $x_i<y_i$ であることを意味する．

を一つずつ直列に，全てが完了するまで実行することが可能であり，それが一つの正当な実行系列を定義する．したがって，$n$ 個の独立な連鎖から成る各タスク・システムに対し，少なくとも $n!$ 個の正当な実行系列が存在する．

閉鎖の問題を取扱うのに重要な定義が二つある．一つは閉鎖の検出に関連して，もう一つは閉鎖の回避に関連して有用である．

**定義 2.3**

$C = C_1 \| C_2 \| \cdots \| C_n$ とし，各 $C_i (1 \leq i \leq n)$ は連鎖であるとする．$\alpha = a_1 \cdots a_k$ を $C$ の部分実行系列とし，$\sigma = s_0 s_1 \cdots s_k$ をそれに対応する部分状態系列とする．連鎖番号の空でない集合 $D$ が存在し，$D$ に属する各 $i$ に対して次式が成り立つと仮定する．

(2.3.3)
$$\mathbf{Q}_i(k) \nleq \mathbf{v}(k) + \sum_{j \notin D} \mathbf{P}_j(k)$$

このとき，$\sigma$ は閉鎖され(てい)る(または，$s_k$ は閉鎖を含む)という．また，$D$ に属する各 $i$ に対して，連鎖 $C_i$ も閉鎖され(てい)る† という††．

この定義は次のことを意味する．すなわち，式(2.3.3)は，$D$ に属する各 $i$ に対し，$\mathbf{Q}_i(k) \neq \mathbf{0}$ であるときにのみ成り立つから，連鎖 $C_i$ で次に生起しようとしている事象があるタスク $T_i(l)$ の開始であるときにのみ成り立つことが分る．さらに，どの $i \in D$ に対しても開始の遷移 $a_{k+1} = \bar{T}_i(l)$ が許容可能でないということを表している．したがって，活動中の連鎖のある空でない部分集合が存在して，これに属していない活動中の連鎖が今後どのような資源の使い方をするかに**かかわりなく**，この部分集合中のどの連鎖も次のタスクの実行を始めることができないならば，閉鎖が存在するということをこの定義は述べている．特に，たとえ閉鎖されていない全ての連鎖が保有資源の全部を解放したとしても，閉鎖されている連鎖のどの一つにとっても，進行するにはなお資源不足であるということが分る．連鎖はどの資源についてもシステムの容量以上に要求することはないという仮定があるので，閉鎖状態には少なくとも二つの連鎖が関与しなければならない．すなわち，$D$ は空でないならば少なくとも二つの連鎖を含むということに注意しよう．

閉鎖されているプロセスの簡単な例が，前述の例の連鎖とシステムに対して，図 2.3-5 に示されている．次の部分実行系列に示されているあらゆる開始の遷移が正当であることは容易に確かめられる．

---

† **訳注**　$C_i$ は‘閉鎖状態にある’ともいう．

†† **訳注**　タスク・システム $C$ も‘閉鎖状態にある’という．

| | $C_1$ q$_1$ | $C_1$ r$_1$ |
|---|---|---|
| $T_1(1)$ | (0, 1, 2, 0) | (0, 0, 2, 0) |
| $T_1(2)$ | (1, 0, 0, 1) | (0, 0, 0, 1) |
| $T_1(3)$ | (0, 0, 2, 0) | (1, 1, 2, 0) |

| | $C_2$ q$_2$ | $C_2$ r$_2$ |
|---|---|---|
| $T_2(1)$ | (2, 0, 2, 2) | (0, 0, 0, 2) |
| $T_2(2)$ | (1, 0, 1, 0) | (3, 0, 3, 0) |

システム： $\mathbf{R} = (R_1 R_2 R_3 R_4)$, $\mathbf{w} = (3, 3, 3, 3)$

| $\alpha$ | | $\bar{T}_1(1)$ | $\underline{T}_1(1)$ | $\bar{T}_2(1)$ | $\underline{T}_2(1)$ | $\bar{T}_1(2)$ | $\underline{T}_1(2)$ | |
|---|---|---|---|---|---|---|---|---|
| P$(k)$ | 0000<br>0000 | 0120<br>0000 | 0100<br>0000 | 0100<br>2022 | 0100<br>2020 | 1101<br>2020 | 1100<br>2020 | 閉　鎖 |
| Q$(k)$ | 0120<br>2022 | 0000<br>2022 | 1001<br>2022 | 1001<br>0000 | 1001<br>1010 | 0000<br>1010 | 0020<br>1010 | |
| | $s_0$ | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ | |
| v$(k)$ | (3333) | (3213) | (3233) | (1211) | (1213) | (0212) | (0213) | |

**図 2.3-5　閉 鎖 の 例**

$$\alpha = \bar{T}_1(1)\,\underline{T}_1(1)\,\bar{T}_2(1)\,\underline{T}_2(1)\,\bar{T}_1(2)\,\underline{T}_1(2)$$

事象 $\underline{T}_1(2)$ の後は，$\mathbf{Q}_1(k) \not\leq \mathbf{v}(k)$ も $\mathbf{Q}_2(k) \not\leq \mathbf{v}(k)$ も成り立つので，$\bar{T}_1(3)$ も $\bar{T}_2(2)$ も共に $\mathbf{v}(k)$ に関して許容可能でない．閉鎖の意味は，もちろん，$C_1$ も $C_2$ もその相手側から強制的に資源を横取りしなければ $\alpha$ の最後の事象よりも先へ進むことはできないということである．

**定義 2.4**

$\sigma = s_0 s_1 \cdots s_k$ が部分実行系列 $\alpha = a_1 \cdots a_k$ に対応する状態系列であるとする．$\alpha$ を語頭として持つ正当かつ完全な† 実行系列が少なくとも一つ存在するならば，$s_k$ は**安全な**(safe)状態であるといい，そうでなければ，それは**安全でない**(unsafe)という．

ある状態が安全であることを証明できるのは，各連鎖の資源の使い方が前もって分っている場合だけであるということは明らかである．閉鎖されたプロセ

† **訳注**　"完全な" 実行系列という用語は未定義である．実行系列の定義自身に，与えられたタスク・システムの**全ての**タスクの開始および終了事象を含むことが明記されているが，このことを強調したものであろう．

スの最後の(閉鎖を含む)状態は明らかに安全でないが，状態は閉鎖されていなくても安全でないことがある．これは図2.3-5の状態 $s_5$ に示されている．この状態からは，実行系列のあり得る(先行制約を満たすような)残りの部分であり，しかも正当かつ完全な実行系列の語尾(suffix)になり得るものはない．実際，一度システムが安全でない状態に入り込むと，安全でない状態にとどまり続けなければならず，閉鎖は避けられない．

閉鎖と資源の状態 $s_k$ を表現する手段として，グラフによる種々の方法が導入されている [5, 6, 7]．本書の目的に対して，グラフを使うことは本質的なことではないけれども，グラフは資源割当てについての形式的なモデルの性質を見易くし，簡単化するのに役立つ．最も簡単なグラフ表現は，資源の型 $R_1$, …, $R_m$ に対応した $m$ 個の節点の集合および次のように定められた枝の集合を持つ有向グラフ $G_k$ を定義することによって得られる．すなわち，$G_k$ に枝 $(R_{j_1}, R_{j_2})$ があるのは，$P_{ij_1}(k)$ と $Q_{ij_2}(k)$ が両方共に零でないような連鎖 $C_i$ が存在するとき，すなわち，$k$ 番目の事象の直後において $C_i$ が型 $R_{j_1}$ の資源を保持しており型 $R_{j_2}$ の資源を要求しているとき，かつそのときに限る．

さて，$s_k$ において，閉鎖状態にある連鎖の集合が $D$ であると仮定する．閉鎖状態にあるから，$D$ の連鎖 $C_{i_1}$ は，$D$ の他の閉鎖された連鎖 $C_{i_2}$ が保持しているある型 $R_{j_1}$ の資源を要求しているはずである．同様に，$C_{i_2}$ は，$D$ の他の連鎖 $C_{i_3}$ が保持しているある型 $R_{j_2}$ の資源を要求しているはずであり，さらに $C_{i_3}$ は $D$ の $C_{i_4}$ が保持しているある型 $R_{j_3}$ の資源を要求している等々である．$G_k$ の定義から，次のように結論することができる．

**命題 2.1**

状態 $s_k$ が閉鎖を含むならば，$G_k$ は閉路(circuit)，すなわち，一つ以上の節点から成る有向の環(loop)(または，輪(cycle))を含む．

$G_k$ の閉路に現れる連鎖† は，与えられた型の資源の全部を必ずしも制御しているとは限らないから，$G_k$ に閉路が存在することは，一般には，閉鎖の存在の十分条件にはならない．たとえば，連鎖 $C$ は，閉路上の他の連鎖が保持している型 $R_j$ の資源を要求しているというだけの理由によって与えられた閉路上にあるのかも知れない．もし $w_j > 1$ ならば，現在は利用可能でない型 $R_j$ の資源を，与えられた閉路上にない他の連鎖がその後解放するかも知れないのである．したがって，$s_k$ では $C$ は必ずしも閉鎖状態にあるとは限らない．し

† **訳注**　$G_k$ の定義からは，連鎖が直接 $G_k$ 上に表現されてはいない．しかし，上の説明で，たとえば，枝 $(R_{j_1}, R_{j_2})$ に連鎖名 $C_{i_1}$ がラベルとして付けられていると考えればよい．

かし，特殊な場合として，$G_k$ の閉路が節点 $R_{j_1}, R_{j_2}, \cdots, R_{j_h}$ から成っていて，これらの型のどの資源も 1 単位しかない $(w_{j_1} = w_{j_2} = \cdots = w_{j_h} = 1)$ ならば，閉鎖が存在するはずであるということは明らかである．

### 2.3.2 予　防

前にも述べたように，必要条件(相互排除，横取り権なし，循環待ち)のうちの一つかそれ以上が成り立ち得ないようにするならば，閉鎖は予防し得る．相互排除の条件は，資源の物理的あるいは論理的な性質から，通常は否定することができない．横取り権なしの条件および循環待ちの条件は，ある特殊な場合には否定することができる．以下にその各々の場合について考察する．

どのような部分実行系列においても，全ての $k$ および全ての連鎖 $C_i$ に対して，$\mathbf{Q}_i(k) \neq \mathbf{0}$ ならば $\mathbf{P}_i(k) = \mathbf{0}$ であるという制約条件が成立するものと仮定する．したがって，式(2.3.3)の右辺はシステムの容量である $\mathbf{w}$ に縮退する．仮定により全てのタスクの開始は $\mathbf{w}$ に対して許容可能であるから，この制約下では閉鎖は起り得ない．この制約は，各連鎖は追加の資源を要求するときにはいつも，資源を全く保持していてはならないということを要請するものであり，単にこれは(閉鎖発生の必要条件の一つであった)横取り権なしを否定したのと同じことになっている．

ある資源を保持している連鎖がそれ以上の要求に応じて貰えないときにはいつも，保持している資源を解放してから追加分の資源と併せて要求するということが義務づけられるならば，上述の制約条件は成立する．この技法の一般的な適用は，あらゆる資源は横取り可能であるという支持できない前提を当然のこととして含んでいる．

上記の制約を実現する別の方法は，各連鎖に，あり得る最大の要求を最初に行うということを認めることである．この方法は，連鎖に割当てられたある資源が長い間使われないままになるかも知れないので，資源の利用率が悪くなるという欠点がある．

しばしば適用される手法として，命題 2.1 に表現されている循環待ちが生じないようにする資源の使い方についての制約を含むものがある [9]．この制約は資源の型についての次のような性質を持つ線形順序(linear order) $\lessdot$ となって現れる．便宜上，資源の型の番号は，$R_1 \lessdot R_2 \lessdot \cdots \lessdot R_m$ となるようにつけられていると仮定する．型 $R_i$ の資源を保持している連鎖が型 $R_j$ の資源を要求できるのは，$R_i \lessdot R_j$ の関係にある場合に限られる．そのような線形順序が存在するときはいつも，連鎖は**順序付資源使用法**(ordered resource usage) の

制約を受けるという.

さて，グラフ $G_k$ に節点 $R_{j_1}, R_{j_2}, \cdots, R_{j_h}$ を含む閉路が存在すると仮定する．もしも，$G_k$ に現れている連鎖が順序付資源使用法の制約を受けているならば，$(R_i, R_j)$ が $G_k$ の枝になり得るのは $R_i \lessdot R_j$ のときに限る．したがって，上で与えられた閉路は $R_{j_1} \lessdot R_{j_2} \lessdot \cdots \lessdot R_{j_h} \lessdot R_{j_1}$ であることを意味しており，線形順序の推移性から次の矛盾が生ずる．すなわち，閉路内の全ての資源の型 $R_{j_i}$ と $R_j$ に対し $R_{j_i} \lessdot R_j$ かつ $R_j \lessdot R_{j_i}$. したがって次の命題を得る.

**命題 2.2**

順序付資源使用法の制約を受ける連鎖の集合の実行に対応する状態系列は，閉鎖されることがない.

入力—実行—出力といった操作の系列におけるように，ある直列の順序で連鎖が資源を必要とするとき，閉鎖を予防する一つの方法としてこの順序付資源使用法が特に適している.

閉鎖を予防する上記の諸方策が部分的にしか適用できないとき，与えられた一つのシステムにおいてそれらを組合わせて使うことも可能であるし，また，設計者がシステム設計時にそれらの制約を具体化するか，あるいはシステムの構成要素としてのタスクをも含む全てのタスク・システムが資源要求時にある種の協約に従うべきことを強制するかのいずれかによって実現可能であることも明らかである.

### 2.3.3　検　　出

状態 $s_k$ が持っている情報を使って，閉鎖が存在するか否かを決定するアルゴリズムを考案できるならば，検出の問題が解けることになる．定義**2.3**に直接に基づいている下記のアルゴリズムは，利用可能な資源量では不足であるような連鎖だけから成る集合を残してその実行を終了するように，$s_k$ における連鎖の実行を全ての連鎖に対してシミュレートしてみることによって，$s_k$ が閉鎖を含むか否かを明らかにするように設計されている.

**アルゴリズム D1**

**段 1.** 初期化: $D \leftarrow \{1, 2, \cdots, n\}$ ; $\mathbf{V} \leftarrow \mathbf{v}(k)$.

**段 2.** $\mathbf{Q}_i(k) \leq \mathbf{V}$ であるような $D$ に属する連鎖番号 $i$ を一つ探す．もし一つも見つからなければ，アルゴリズムは終了する.

**段 3.** $D \leftarrow D - \{i\}$ ; $\mathbf{V} \leftarrow \mathbf{V} + \mathbf{P}_i(k)$; 段 2 へ行く.

アルゴリズム D1 についてなされるべき重要な観察は，利用可能資源ベクトル $\mathbf{V}$ はアルゴリズムのパス（段 2，3 の実行）の継続に対して常に単調非減少であることであり，これは，（実際，二つ以上の系列が正当である場合）実行可能な連鎖が完了する順番に関わっていない．したがって $i$ 回目のパス（パス $i$）で実行可能な任意の連鎖は，任意の $j > i$ に対して，パス $j$ でも実行可能でなければならない．逆に，利用可能資源ベクトルの最終の値は，（最終の）集合 $D$ に属するどの連鎖に対しても不足しているから，この集合は正確に閉鎖された連鎖の集合であることになる．このアルゴリズムは，全ての連鎖が完了できるかどうかを調べるのではなく，各連鎖のタスクがもう一つ完了できるかどうかだけを調べるものであることに注意しよう．

アルゴリズム D1 は，最悪の場合には全てのパスにわたって集合 $D$，すなわち，パス $i$ においては $(n-i+1)$ 個の連鎖から成る集合の全要素を探さなければならない．したがって，このアルゴリズムの走行時間は，$m\sum_{i=1}^{n}(n-i+1) = \mathcal{O}(m\cdot n^2)$ に比例する†．資源の状態に対してもっと凝った表現法を使うと，走行時間が $\mathcal{O}(m\cdot n)$ である検出アルゴリズムを作ることができる [6]．

この速い方のアルゴリズムの原理は次の通りである．利用可能資源ベクトル $\mathbf{V} = (V_1, \cdots, V_m)$，現在の要求行列 Q および現在の資源割当て行列 P が与えられているとする．各列 $j$ に対し，$Q_{ij} \leq V_j$ ならば欄 $Q_{ij}$ に印をつける．行 $i$ の全ての欄に印がつけられると，$\mathbf{Q}_i \leq \mathbf{V}$ であることになり，連鎖 $C_i$ の次のタスクの開始が $\mathbf{V}$ に関して許容可能となる．$C_i$ は究極的には完了するものと仮定すると，現在保持している資源 $(\mathbf{P}_i)$ は究極的には返して貰えるから，新しい利用可能資源ベクトル $\mathbf{V}' = \mathbf{V} + \mathbf{P}_i$ を用いて Q 内の欄に印をつける作業を続けることができる．$Q_{ij} \leq V_j \leq V_j'$ であるから，前に印をつけた全ての欄は印がついたままでよい．この過程を続けると究極的には，$\mathbf{Q}$ の全ての欄に印がつくか，または，印のついていない欄を少なくとも一つ含むような行の番号の集合 $D$ が残るかのいずれかである．後者の場合，各 $j \in D$ に対して，$\mathbf{Q}_j$ に応ずる開始の遷移は，

$$\mathbf{V} + \sum_{i \notin D} \mathbf{P}_i$$

に関して許容可能でない．したがって，$D$ に属する番号を持つ連鎖は閉鎖されていることを意味する．

印をつける手続きを能率化するため，手続き中の任意の時点で，Q の行 $i$ 内

---

† 記法 $\mathcal{O}(f(n))$ は "$f(n)$ の位数" と読み，大きさが，全ての $n$ に対して $|f(n)|$ のある定数倍以下である量を表す．

の印がついていない欄の数 $x_i$ を要素とするベクトル $\mathbf{x} = (x_1, \cdots, x_n)$ を定義する．最初は，$\mathbf{x} = (m, \cdots, m)$ である．$Q_{ij}$ に印をつけたときにはいつも $x_i$ から1を引き，$x_i = 0$ になると $\mathbf{V}$ に $\mathbf{P}_i$ を加える．この手続きをさらに改善するためには，列 $j$ でまた印がつけられていない $Q_{ij} \leq V_j$ をもっと効率良く探す必要がある．このため，列 $j$ の欄を，$Q_{i_1j} \leq Q_{i_2j} \leq \cdots \leq Q_{i_nj}$ になるように待ち行列

$$E_j = (Q_{i_1j}, Q_{i_2j}, \cdots, Q_{i_nj})$$

の形に並べ替える．列 $j$ の探策手続きには，$E_j$ の先頭の欄 $Q_{hj}$ が，$Q_{hj} \leq V_j$ であるかどうかの判断が含まれる．もしそうならば，$x_h$ から1を引き，$E_j$ から $Q_{hj}$ を消去する．明らかに，この探索アルゴリズムの走行時間† は待ち行列 $E_1, \cdots, E_m$ を空にするのに要する時間 $O(mn)$ に比例する．このアルゴリズムの形式的な仕様を下に示す．便宜上，任意の列 $\omega = z_1z_2 \cdots z_r$ に対し，**head** $(\omega) = z_1$, **tail** $(\omega) = z_2 \cdots z_r$ と定義する．

**アルゴリズム D2**

$D \leftarrow \{1, 2, \cdots, n\}$ ; $\mathbf{V} \leftarrow \mathbf{v}(k)$; $\mathbf{x} \leftarrow (m, \cdots, m)$; *done*←**false**;
**while not** *done* **do**
　　*done*←**ture**;
　　**for** $j = 1, 2, \cdots, m$ **do**
　　　**while** $E_j$ nonempty **and head**$(E_j) \leq V_j$ **do**
　　　　$h \leftarrow$ (**head**$(E_j)$ の要求を出した連鎖の番号)
　　　　$E_j \leftarrow$ **tail**$(E_j)$
　　　　$x_h \leftarrow x_h - 1$
　　　　**if** $x_h = 0$, **then**{$\mathbf{V} \leftarrow \mathbf{V} + \mathbf{P}_h(k)$; $\mathbf{D} \leftarrow \mathbf{D} - \{h\}$ ; *done* ←**false**}

アルゴリズム D2 が停止したとき，集合 $D$ は(空でなければ)閉鎖された連鎖の番号を含んでいる．実際には，いくつもの待ち行列 $E_j$ から欄を消去する必要はなく，その代りに，各リスト $E_j$ 内を指す一組のポインタを，探索の進行に連れて進むように維持すればよい．

$E_1, \cdots, E_m$ が与えられたときには，アルゴリズム D2 を実行する時間は $mn$ に比例するけれども，タスク・システムの実行に伴って $E_1, \cdots, E_m$ を要求量の昇順に維持するための隠れた費用がある．実際問題として，これらのソ

† **訳注**　待ち行列 $E_j$ の作成時間は含まない．

ートされた待ち行列を維持するならば，要求行列を別の項目として保持する必要はない．アルゴリズム **D2** が余分に複雑になった埋め合せとして，ソートされた要求の待ち行列は，資源が解放された結果，待機中の連鎖に利用可能になる度ごとに，どのタスクを活性化できるかを決定する方法を簡単にしてくれるという大きな利点を持つ．特に，利用可能になった資源に対応する待ち行列を走査し，今や要求が満たされるようになった全ての連鎖 $C_i$ に対して計数 $x_i$ を減らすことが必要になるだけである．減少して零になった計数を持つ任意の連鎖は，この時点で次に実行するべきタスクを開始させ得る．

閉鎖を検出する機構が与えられたとき，閉鎖状態から回復させる最も簡単な手法としては，恐らく，閉鎖された連鎖の各々を強制終了させる方法とか，それ程荒っぽくはないが，閉鎖が除去されるのに十分な資源量になるまで，閉鎖された連鎖をある順番で強制終了させていく方法などがある．明らかに，もし強制終了されるならば閉鎖が除去されるであろうような連鎖の極小の集合を探すアルゴリズムも設計できる．強制終了されつつある閉鎖状態のタスクから型 $R_j$ の資源を **1** 単位取り除く(強制的横取りの)度ごとに，一定の費用 $C_j$ を計上するもっとも一般的な手法が工夫されている[8]．この回復のアルゴリズムは，強制終了に要する総費用が最小となるタスクの集合の探索を行う．

### 2.3.4 回　避

本書での回避の手法の定義によると，連鎖による資源の使い方についての予知情報が必要である．この情報は，各タスク $T_i(l)$ の要求ベクトル $\mathbf{q}_i(l)$ および解放ベクトル $\mathbf{r}_i(l)$ の具体的指定という形式をとる．

閉鎖を回避するためのアルゴリズムの機能は，システムの状態が常に安全であるように事象の生起順序を制御することである(定義 **2.4** 参照)．したがって，そのようなアルゴリズムは，連鎖が資源を要求あるいは解放する度ごとに呼ばれなければならない．状態 $s_k = [\mathsf{P}(k), \mathsf{Q}(k)]$ が安全であることが分っており，回避のアルゴリズムは遷移 $s_k \to s_{k+1}$ が許容可能かつ安全である† かどうかを決めようとしているものと仮定する．二つの場合，すなわち，ある $i$ と $l$ に対して $a_{k+1} = \underline{T}_i(l)$ の場合と $a_{k+1} = \bar{T}_i(l)$ の場合があり得る．前者の場合，状態 $s_{k+1}$ は次式で定義される．

---

† **訳注**　この表現はこれまでの用語法からやや逸脱している．まず，これは「遷移 $s_k \to s_{k+1}$ が許容可能であり」かつ「状態 $s_{k+1}$ が安全である」の意であり，さらに，遷移 $s_k \to s_{k+1}$ が終了の遷移である場合にはいつも許容可能であると見なし，それが開始の遷移である場合にはこれまでの定義に従うのであると理解しよう．

$$\mathbf{P}_i(k+1) = \mathbf{P}_i(k) - \mathbf{r}_i(l) \qquad \mathbf{Q}_i(k+1) = \mathbf{q}_i(l+1) \quad (l < p_i)$$
$$\mathbf{P}_h(k+1) = \mathbf{P}_h(k) \qquad \mathbf{Q}_h(k+1) = \mathbf{Q}_h(k) \quad (h \neq i)$$
$$\mathbf{v}(k+1) = \mathbf{v}(k) + \mathbf{r}_i(l)$$

$\mathbf{v}(k) \leq \mathbf{v}(k+1)$かつ$s_k$が安全であったから，状態$s_{k+1}$も安全である．たとえ終了の遷移が許容可能かつ安全であるとしても，回避のアルゴリズムはここで重要な役割を演ずる．というのは，利用可能な資源が増加し，それ以前は許容可能でなかった一つ以上の開始の遷移を許容可能にするかも知れないからである．

後者の場合，$\mathbf{q}_i(l) \leq \mathbf{v}(k)$ならば$s_{k+1}$への遷移は許容可能で，$s_{k+1}$は次式で定義される．

$$\mathbf{P}_i(k+1) = \mathbf{P}_i(k) + \mathbf{Q}_i(k) \qquad \mathbf{Q}_i(k+1) = 0$$
$$\mathbf{P}_h(k+1) = \mathbf{P}_h(k) \qquad \mathbf{Q}_h(k+1) = \mathbf{Q}_h(k) \quad (h \neq i)$$
$$\mathbf{v}(k+1) = \mathbf{v}(k) - \mathbf{Q}_i(k)$$

たとえ$s_{k+1}$へのこの遷移が許容可能であったとしても，$s_{k+1}$は安全ではないかも知れない．したがって，回避のアルゴリズムは，開始の遷移直後の状態の安全性を検査する手続きを組込んでおかなければならない．

一般に，安全性についての問に回答する手続きは，活動中の連鎖に対する未生起の事象から成り，しかも実行系列の語尾となり得る全ての系列を数え上げる探索法を使うことになる．そのような系列を**完了系列**(completion sequence)という．この探索手続きの目的は，$\mathbf{v}(k+1)$が最初に与えられた利用可能な資源であると仮定して，開始の遷移としては許容可能なものだけを含むような完了系列を一つ得ることである．

完了系列は実行系列の任意の語尾である．$a_1 \cdots a_k a_{k+1} \cdots a_{2r} (r = p_1 + \cdots + p_n)$が連鎖$C_1, \cdots, C_n$の並列結合の実行系列であるならば，$a_{k+1} \cdots a_{2r}$は状態$s_k$に対する完了系列である．完了系列の任意の語頭を**部分完了系列**という．部分完了系列は，その中にあるあらゆる開始の遷移が許容可能であるならば，**正当である**(valid)という．次の諸結果は，効率良く正当な完了系列を探す手続きを工夫するときに役に立つ．

**命題 2.3**

$s_k$は安全で，($\beta$における$C_i$以外への全ての割当てが安全であり†,)かつ，

† **訳注**　この(　)内の部分は，原著の初刷では欠落している．この“割当てが安全”という文脈は曖昧である．

$a_{k+1} = \bar{T}_i(l)$は許容可能であると仮定する．$\beta\underline{T}_i(p_i)$が $s_{k+1}$ に対する正当な部分完了系列であるならば，$s_{k+1}$ は安全である†．

---

† **訳注**　訳著等にとってこの命題を証明することは不可能である．そこで命題2.3，その証明，およびその後に続くこの命題の説明に取って代るものを以下に付加することにする．

**[命題 2.3-1]**　$\alpha\bar{T}_i(l)$は安全な状態 $s_k$ に対する正当な完了系列の語頭であり，しかも $\bar{T}_i(l)$ による遷移 $s_k \to s_{k+1}$ は許容可能であるとする．このとき，$\alpha$ のある置換 $\beta$ が存在して，これが $s_{k+1}$ に対する正当な部分完了系列であるならば，$s_{k+1}$ は安全である．

**証明**　$s_{k+1}$ は安全でないと仮定しよう．

命題の条件から，ある系列 $\gamma$ が存在して，$\alpha\bar{T}_i(l)\gamma$ は $s_k$ に対する正当な完了系列になる．$s_{k+1}$ は安全でないので，システムの先行制約を満たすいかなる $\alpha$ の置換 $\alpha'$ に対しても，$\alpha'\gamma$ は $s_{k+1}$ に対する正当な完了系列とはなり得ない．

さて，状態 $s_k$ からは，系列 $\alpha\bar{T}_i(l)$によって状態 $s_j$ に遷移し，系列 $\bar{T}_i(l)\beta$によって状態 $s_j'$ に遷移するものとする．$s_{k+1}$ に対して，$\beta$ は正当であるけれども，$\beta\gamma$ は正当ではないから，$s_j'$ に対して $\gamma$ は正当でないことになる．しかし，明らかに $s_j = s_j'$ であるので，このことは $s_j(=s_j')$に対して $\gamma$ が正当であることと矛盾する．

したがって，$s_{k+1}$ に対して $\alpha$ の置換である正当な部分完了系列が存在するならば，$s_{k+1}$ は安全でなければならない．　■

$s_k$ が安全であることを知っているということは，少なくとも $s_k$ に対する正当な完了系列の一つ(その語頭が $\alpha\bar{T}_i(l)$であるとする)を知っていることを意味すると考えてもよいであろう．この前提条件の下で，$s_{k+1}$ の安全性を知りたいとき，命題 2.3-1は，$s_{k+1}$ に対する完了系列に比較して長さの短い $\alpha$ の置換に対する探索のみによって $s_{k+1}$ の安全性について結論を効率よく下せる場合があるということを述べている．その代り，この命題は十分条件だけを与えているので，これだけでは結論を下せない場合もある．しかし，以下に述べる事柄はある程度重要であると思われる．まず，$\alpha$ は連鎖 $C_i$ の事象を含まないことに注意しよう．今，システム内の連鎖数 $n$ は 2 であるとする．この場合には，$\alpha' \neq \alpha$ であるような $\alpha'$ はあり得ないから，$s_{k+1}$ に対して $\alpha$ だけを調べればよい．$s_0$ に対して連鎖の直列実行は常に正当であることを知っているから，これから出発して 2 連鎖システム $C_1 \| C_2$ の閉鎖を生じない並列実行のための事象の系列が効率よく求められる場合があり得る．

さらに，[命題 2.3-1] の拡張形を二つ次に示そう．

**[命題 2.3-2]**　$\alpha\delta$ は安全な状態 $s_k$ に対する正当な完了系列の語頭であり，しかも $\delta$ は $s_k$ に対する正当な部分完了系列でもあって，状態 $s_k$ は $\delta$ によって $s_j$ に遷移するものとする．このとき，$\alpha$ のある置換 $\beta$ が存在して，これが $s_j$ に対する正当な部分完了系列であるならば，$s_j$ は安全である．

**証明**　省略する．　■

図 2.3-4 に示されている正当な実行系列の例は，$C_1$ と $C_2$ の直列実行の系列からこの命題の応用によって求められる．

**[命題 2.3-3]**　$\alpha\beta$ は安全な状態 $s_k$ に対する正当な完了系列の語頭であり，しかも $\beta$ は $s_k$ に対する正当な部分完了系列であるとする．$\beta$ に対して集合 $A, B$ を次のように定義する．

$$A = \{(i, l) | \bar{T}_i(l)\text{は }\beta\text{ 中に出現する開始の事象}\}$$
$$B = \{(i, l) | \underline{T}_i(l)\text{は }\beta\text{ 中に出現する終了の事象}\}$$

このとき，$C \subseteq A - B$ を満たす任意の $C$ に対して

$$\sum_{(i,l) \in A} \mathbf{q}_i(l) \leq \sum_{(i,l) \in B \cup C} \mathbf{r}_i(l)$$

が成り立つならば，$(i, l) \in C$ であるような終了事象 $\underline{T}_i(l)$の全てから成る任意の系列 $\beta'$ に対して，$\beta\beta'\alpha$ はある正当な完了系列の語頭になる．

**証明**　読者にまかせる．なお，後出の命題 2.5 の証明を参考にせよ．　■

この命題 2.3-3 は本文後出の命題 2.4 を包含している．命題 2.5 との関連については読者の一考を煩わしたい．

これを証明するために，$s_{k+1}$ が安全でないと仮定する．$s_k$ は安全であるから，要求 $\mathbf{q}_i(l)$を認めることができ，しかもそれは閉鎖された連鎖の一つに $C_i$ を含むような状態に不可避的に至らしめることになる．しかし，部分完了系列 $\beta\underline{T}_i(p_i)$は，$C_i$ が閉鎖されないことを示している．

これに関連したもう一つの結果として，任意の状態 $s_k$ に対し，資源を保持している全連鎖が閉鎖されることなしに完了できるならば $s_k$ は安全であるという命題がある．明らかに，これらの連鎖が完了した後に，残りの連鎖は常に任意の(直列な)順序で実行することができる．この結果を次の形で述べる．

**命題 2.4**

$A = \{C_i|\mathbf{P}_i(k) > \mathbf{0}\}$は，状態 $s_k$ において資源を保持している連鎖の集合であるとする．$\beta$ が，$A$ に属する全ての $C_i$ に対する終了事象 $\underline{T}_i(p_i)$を含む正当な部分完了系列であるならば，$s_k$ は安全である．(したがって，正当な完了系列の探索は，資源を保持している連鎖内の事象に限定することができる．)

探索時間は，次の結果 [5] を用いてさらに改善することができる．

**命題 2.5**

$s_k$ は安全な状態であるとする．$\beta$ を $s_k$ に対する正当な部分完了系列であるとし，$\beta$ 内で開始される全てのタスク $T_i(l)$に対し，$\mathbf{q}_i(l) \leq \mathbf{r}_i(l)$であると仮定する．このとき，$\beta$ はある正当な完了系列の語頭である．

**証明**　$s_k$ が安全であると仮定すると，ある正当な完了系列 $\theta$ が存在する．$\beta$ 内で開始されるけれども終了はしないタスクに対する $\theta$ 内の全ての終了事象を $\beta$ の後に(任意の順序で)付加して得られる系列を $\beta'$ とする．$\beta$ は正当であり，終了事象はいつも許容可能であるから，$\beta'$ も正当でなければならない．$\theta$ から $\beta'$ に含まれる事象を全て除いて得られる系列($\theta$ に関する $\beta'$ の補系列)を $\beta''$ とする．さて，命題の条件の下で，$\theta' = \beta'\beta''$(明らかに $\beta$ を語頭として含む)は正当な完了系列である．これを証明するには，$\beta'$ が正当であることは分っているから，残りの $\beta''$ も正当であることを確かめればよい．

$\bar{T}$ を $\beta''$ 内のタスクの開始事象であるとする．$\theta$ 内で $\bar{T}$ より先行する事象は全て $\theta'$ 内でも $\bar{T}$ に先行するが，一般には，$\theta$ 内で $\bar{T}$ に後続するその他のタスクの開始と終了の事象もまた，$\theta'$ 内で $\bar{T}$ に先行することがある．しかし，$\beta'$ の定義から，そのような開始 $\bar{T}'_i$ の各々に対してはいずれも，少なくとも $\bar{T}'_i$ で獲得した資源以上の資源を解放する終了事象 $T'_i$ が $\beta'$ 内に後続していなければならない．すなわち，$\mathbf{q}'_i \leq \mathbf{r}'_i$ が $T'_i$ の要求/解放ベクトルとして成り立

っているはずである．また，$\theta'$ 内では $\bar{T}$ に先行するけれども $\theta$ 内ではそうではないようなその他のタスクの終了はいずれも，$\theta'$ においては $\bar{T}$ にとって利用可能な資源を増加させ得るだけであることに注意しよう．したがって，$\theta'$ において $\bar{T}$ の直前に利用可能な資源の量は，少なくとも $\theta$ において $\bar{T}$ の直前に利用可能な資源の量はなければならない．■

この結果の意味するところは分り易く説明できる．$s_k$ に対する正当な完了系列を探すとき，開始時に必要とした量以上の資源を終了時に解放するようなタスクの開始のみから成る最長の正当な部分完了系列 $\beta$ を見つけるようにするべきである．特に，$\beta$ で始まるある完了系列が正当でないことが分ったとしても，探索手続きを逆戻りさせて $\beta$ を語頭として持たない他の完了系列を調べる必要はない．

完了系列の探索を改善するために概説してきた諸技法にもかかわらず，本質的に数え上げ法ではないアルゴリズム(たとえば，実行時間が完了系列中の事象の数の指数関数になるのではなく，多項式として表すことができるアルゴリズム)を設計することはできない [7]．これまでは，資源の使い方に関する完全な予知情報があることを仮定した閉鎖回避のモデルを扱ってきたから，そのようなアルゴリズムをこれ以上考察しない．実際問題として，これは一般には実現可能な仮定ではない．もしかするとプログラマにとって大変な重荷になるかも知れないというだけではなく，資源の使い方が計算開始後のある時点まで分らないという場合(たとえば，資源の使い方がデータに依存している場合)には，全く不可能な仮定である．したがって，資源の使用量の最大(または上界)が予め知られていることが仮定されているようなモデルの方がより実際的である．このようなモデルを **MRU(最大資源使用限度——maximum resource usage)システム**という [10]．

MRU システムについての形式的な取扱いでは，各連鎖 $C_i$ に関連して予め知られている情報は，$C_i$ がいつかは必要とすることになる $R_j$ の最大量を $M_{ij}$ とするとき，ベクトル $\mathbf{M}_i = (M_{i1}, \cdots, M_{im})$ だけであるという仮定の下に，閉鎖を回避する問題を考察する．$C_i$ のタスクによって表現すると，$\mathbf{M}_i$ は次式で与えられる．

$$M_{ij} = \max_{1 \leq l \leq p_i} \{q_{ij}(l) + \sum_{h=1}^{l-1} (q_{ij}(h) - r_{ij}(h))\}$$

今や，資源の使い方についての部分的な情報しかないので，閉鎖の回避を保証

するためには，より制限された安全状態の概念を考慮しなければならない．特に，MRU システムでは，連鎖のタスクの構成が実行してしまうまで分らないという事実に考慮を払わなければならない．

以上を念頭において MRU システムにおける回避問題を考察しよう．システムが状態 $s_k$ にあり，Q$(k)$内のある要求を叶えても安全であるかどうかを決定するために吟味中であるとする．前と同様に，もしその要求を叶えたとすれば，新しい状態 $s_{k+1}$ は，ある $i$ に対して $\mathbf{Q}_i(k+1)=\mathbf{0}$ および $\mathbf{P}_i(k+1)=\mathbf{P}_i(k)+\mathbf{Q}_i(k)$ となる点でだけ $s_k$ と異なることになる．$s_k$ が閉鎖を回避できるような状態であると仮定したとき，どのような条件があれば(仮の)新しい状態 $s_{k+1}$ もこの性質を持っているといえるかを決定したい．それが満たされるならば吟味中の要求を実際に叶えることができる．

この問題は，最大必要量 $\mathbf{M}_i$ しか分らないという制約の下に解かなければならない．したがって，閉鎖を $s_{k+1}$ について回避することができることを絶対的に保証するためには，活動中の連鎖内に残っているどのようなタスクに対しても，可能な限りの最悪(最多)量の資源が必要になることを想定しなければならない．最悪の場合というのは，活動中の各連鎖 $C_i$ の残りの部分が $s_{k+1}$ で[すなわち，$\mathbf{P}_i(k+1)$ で]割当てられている資源を一つも解放せず，引続いて完了前にその最大割当てを必要とするときに，起ることになる．さて，$C_i$ がその最大必要量を取得するには，$\mathbf{M}_i-\mathbf{P}_i(k+1)$ 個の資源を追加要求しなければならない．したがって，閉鎖の回避についての決定を行うことに関する最悪の場合は，次のような簡単な問題としてモデル化できる．すなわち，$s_{k+1}$ において活動中の連鎖を $C_{i_j}(j=1,\cdots,a)$ と表す．資源ベクトルの初期値を $\mathbf{v}(k+1)$ として，(独立な)タスク $T'_1,\cdots,T'_a$ に対する要求および解放ベクトルが次式で与えられるとき，$T'_1,\cdots,T'_a$ の正当な実行系列を見出せという問題である．すなわち，

$$
(2.3.4)\qquad \begin{aligned} \mathbf{q}'_j &= \mathbf{M}_{i_j}-\mathbf{P}_{i_j}(k+1) \quad (1\le j\le a) \\ \mathbf{r}'_j &= \mathbf{M}_{i_j} \end{aligned}
$$

明らかに，タスク $T_j'$ は連鎖 $C_{i_j}$ の残りの部分のモデルになっている．$C_{i_j}$ には，タスクへの分解と定義により一般には複数個のタスクが残っている．しかし，最悪の場合という仮定から最大量 $M_{i_j}$ は一度に獲得され，しかも($C_{i_j}$ の完了までは)少しも解放されないので，引続いて要求されることはない．したがって，残りのタスクは唯一つだけであるという仮定は，資源が連鎖から解放されるか，新しい資源が要求されるか，あるいはその両方であるかの場合に

のみ新しいタスクが現われ得るという要請に整合している.

定義2.3とアルゴリズムD1, D2から, 次の結果を得る.

**命題 2.6**

状態 $s_{k+1}$ が与えられたとき, 次のように定義する.

$$\begin{aligned} \mathbf{Q'}_i(k+1) &= \mathbf{M}_i - \mathbf{P}_i(k+1) \qquad && (\mathbf{P}_i(k+1) + \mathbf{Q}_i(k+1) > 0 \text{ のとき}) \\ &= 0 && (\text{そうでないとき}) \\ & && (1 \leq i \leq n) \end{aligned}$$

また, $\mathrm{Q}'(k+1)$は列 $\mathbf{Q'}_i(k+1)$ $(1 \leq i \leq n)$から成る行列であるとする. 状態 $s_{k+1}$ は, 状態 $[\mathrm{P}(k+1), \mathrm{Q}'(k+1)]$が閉鎖を含まないならば, 定義2.4から, 安全である.

先程の議論とアルゴリズムD1(または, 状態を適当に構成するならばD2)の性質により, 状態 $[\mathrm{P}(k+1), \mathrm{Q}'(k+1)]$は, 上記の式(2.3.4)で定義されたタスク $T'_1, \cdots, T'_a$ の正当な実行系列を見つけることができないときかつそのときにのみ閉鎖を含む. このことは, タスク $T'_j$ が最悪の場合を模倣するように作られているので, 安全な状態に対する一つの十分条件が得られたことを意味する. しかし, 部分的な情報しか使わないことの代償は, それが必要条件ではないことである. したがって, 一般に, 最悪の場合には閉鎖を引き起すことになるという理由によって要求を拒否(延期)することになるけれども, 実際にはその要求を叶えても必ずしも閉鎖が不可避になるとは限らない.

その一方で, 閉鎖回避の手続きの効率(複雑さ)が著しく改善されている. 特に, 命題2.6から, 走行時間が $n$ に関して線形であるアルゴリズムD2を使って閉鎖を回避し得るからである.

## 2.4 相 互 排 除

### 2.4.1 はじめに

本節と次節の議論は, プロセスの制御と同期を実現するためのプログラミングの道具に関するものである. タスクは次の二通りのあり方で相互に作用し合う. すなわち, 間接的には同じ資源を競合することによって, また, 直接的にはメッセージを送ったり待ったりすることによってである. 閉鎖問題でも見てきたように, 独立なタスクが実行中であるとき, それらの進行を制御する責任はそれらを下で支えているシステムに掛かっている. しかし, タスクが直接に

相互作用を及ぼし合うとき，互いの進行を制御する機構はそれらのタスクをプログラム化する際に明白に表現されていなければならない．これから議論する相互排除と同期のための諸道具は後者の型† のものである．

相互排除または調停の問題は，二つ以上のタスクによって同時に使用されないように再使用可能な資源†† を制御することに関係している．そのような資源の例として，プロセッサ，記憶のページ，行単位の入出力装置からデータベースなどまでがある．タスクの正しい動作という点から見ると，タスクは資源の使用中にその状態を変えるから，一時には高々一つのタスクにしか資源を与えないようにすることが必要である．しかし，タスクは資源を使う前にその状態を初期化するので，資源を使うタスクの順番は重要でない．それゆえに相互排除問題では，資源を使うタスクの順番に予め制約が課せられているような解はどのようなものであっても望ましいものではない．単にそれだけではなく，与えられた資源をどのタスクが使うことを望んでいるかとか，タスクが資源を要求したときにその型の資源は何単位利用可能であるのかとかを予めいうことは不可能であるから，このような状況の下では，資源の効率的な使い方を保証するように予めタスク間に先行制約を課することすらできないであろう．

資源の相互排他利用を実現することは，もしも資源の使い方が一つのオペレーティング・システムの集中的管理下にあるならば実際には問題ない．というのは，その場合，タスクの設計という観点から考慮すべき問題は，特定の資源を要求するかまたは解放することをオペレーティング・システムに知らせるためにタスクが守るべき規約についての問題だけであるからである．そのような規約については後程さらに言及するが，今のところは，タスク間通信のための公共的(大域的，globally)に到達可能な変数を用いてタスク自身が相互排除を実現しなければならないものと仮定しておく．互いに排他的に実行されるべきタスク(あるいは，より正確にコードの一部分)は，諸文献では**際どい部分**(critical section)と呼ばれている．

相互排除問題に対する望ましい解を見出すことは，起り得る競合条件(race condition)および別々のプロセッサ上で実行中のタスクの(有限かつ非零の)速度の変動のため，明らかに自明ではない．したがって，たとえば，異なるプロセ

† Wirth[11]は並行性を表現する2種類のプログラミング言語のしかけを提案している．存在しているタスク間の直接的な相互作用を取扱うしかけのほか，プログラムやコードのある部分が潜在的には異なったタスクで実行中になり得ることを宣言するためのしかけを提案したのである．この種の言語学上の機構は主としてコンパイラやオペレーティング・システムに，資源の割当てを決定する際により大きな融通性を与えることが意図されている．

†† 内部状態が使用中は変更されるけれども，他のタスクに割当てる際には初期化することができるような資源．

ッサ上に二つのタスク $T_1$ と $T_2$ があって，どちらも相手側が資源 $R$ を使うことを阻止する十分な機会を持たないうちに，双方ともに欲しい資源 $R$ が利用可能であることを認識して動作してしまうかも知れないという可能性に注意を払わなければならない．たとえば，この問題を解こうとして，$R$ の利用可能性を相手側に知らせ合うために，二つのタスク $T_1$ と $T_2$ によって使われる一つの公共的変数を定義したとするならば，この可能性が問題となる．つまり，$T_1$ がこの変数を調べて $R$ が利用できることを知った後，その値を $R$ が使用中であることを示す値に $T_1$ が変更する前に，$T_2$ も同じことをする可能性がある．このような公共的変数に到達するための主記憶参照間の調停は，ハードウェアのレベルの相互排除の一例であることに注意しよう [12]．すなわち，いくつかの要求が同時に到着するかも知れないけれども，ハードウェアはある順番で一時に一つずつサービスする．

本節の相互排除問題の考察では，議論を簡単にするため，一時に高々一つのタスクでしか使用できない 1 個の資源 $R$ のみが存在するという仮定を置こう．複数個の資源があることを考慮に入れたより一般的な取扱いは下記の解析を拡張することによって容易に得ることができる．

以前に定義した記法を用いて相互排除を容易に形式化することができる．

**定義 2.5**

$\alpha = a_1a_2 \cdots a_k$ をタスク・システム $C = (\mathcal{T}, \lessdot)$ のある部分実行系列とする．タスク $T$ は，ある $i \leq k$ に対して $a_i = \bar{T}$ かつ全ての $j(i < j \leq k)$ に対して $a_j \neq \underline{T}$ であるならば，$\alpha$ の後で**活動中**(active)であるという．タスク $T$ および $T'$ は，$\alpha$ のどの語頭の後でも，$T$ と $T'$ のうちの高々一つが活動中であるとき，かつそのときのみ，$\alpha$ において**互いに排除されている**(mutually excluded)という．

明白な理由で，相互排除の実現は次のような付帯条件を満たすことが望ましい．それは問題をかなり複雑にし，それ程容易には形式化されない．ごく簡単に言えば，その条件は，$R$ の使用を要求している個々のタスクが，$R$ を要求または使用している他のタスクによって無定限に阻止されないことである．これを言い換えると，我々の解はどのような状況の下でも，次のような意味で中身のないものになってはならない．すなわち，$R$ を必要とする一つ以上のタスク(それがどのようなものであっても)による $R$ の使用を他のタスクが阻止することによって相互排除が達成され，その結果，あるタスクの進行が無定限に阻止されることがあってはならない．望ましい解は，たとえば，タスクが“お

先にどうぞ”式のループ内に無定限に留まるような事態を生じてはならない．これは，閉鎖について前節で導入した安全性の概念に関連するから，無定限の阻止が避けられているとき，その解は**安全である**(safe)ということにしよう．

有限のシステムと実行系列を扱っているので，循環的タスク・システムを考えるのでなければこの条件が満たされることは直ちに分る．循環的タスク・システムの場合は，“無定限に阻止される”ことについて次のような解釈を採用する．次式に示す循環的システムの実行系列を考えよう．ただし，$n_i (1 \leq i \leq k)$を$C_i$内のタスク数とする．

(2.4.1) $$C = C_1^{m_1} \parallel C_2^{m_2} \parallel \cdots \parallel C_k^{m_k}$$

このとき，無定限の阻止がないこと(安全条件)は，任意のシステム$C_i$内のどのタスクも$R$の使用を待って費す時間が$k$個の数$n_j (1 \leq j \leq k)$のみを変数とする任意の関数によって定まる有限値で押えられることを意味すると考えよう(待ち時間とは，循環的に実行しているタスクの終了と次回の開始との間の実行系列内に現れる事象の数を意味する)．与えられた$C_i (1 \leq i \leq k)$の具体的指定に対してこのことは特に，$m_i$がどのように大きくても任意のタスクが阻止される時間はある一定の定数より小さいことを意味する．

一般の問題を取扱う前に，ある制限的な仮定がなされたときの相互排除を実現する単純化された技法を吟味しよう．この取扱い方は非形式的で，一般の問題における，より微妙な様相に関心を向けるのに役立つであろう．

### 2.4.2 相互排除を実現するための基本操作

もしも公共的変数に対するいくつかの操作が不可分である(すなわち，任意の時点でそれらの中の高々一つが実行中であり得て，かつ実行中の操作は他の操作が始まる前に完了しなければならない)と仮定することができるなら，相互排除を実現するタスクの設計はかなり簡単になる．一例として，相互排除を実現するために Knuth [13] が考案した機構を示そう．その機構では，不可分であることが仮定される操作は，記憶に対する1回の読取りまたは1回の書込み(一般の問題においては公共的変数に対する不可分操作はこれだけである)に比べると遥かに巧く作られている．そこでは，相互排除の問題を完全に解決する代りに，その問題のかなりの範囲をハードウェアの設計者に手渡し，必要な不可分操作の相互排除を実現する責任を負わせている．

タスク$T_1, \cdots, T_n$から成るタスク・システム$C$を考え，それらのうち少なくとも二つのタスクが資源$R$を必要とするという理由で，互いに排除されなけ

ればならないものと仮定する．これらのタスクの相互排除が保証されるように $C$ を部分的に修正しよう(明らかに，$R$ を使うタスクのどの対も互いに独立でないなら，既に相互排除は保証されている)．特に，$R$ を使う各タスク $T_i$ に対し，新しいタスク $S_i$ と $U_i$ を導入し，それぞれ $T_i$ の直前と直後に置くものとする．この部分修正は図 2.4-1 に示されている．$T_i$ は($R$ を使うことが知られていることを除いて)解釈されないままであるけれども，$S_i$ と $U_i$ は完全に仕様が定められる．この $S_i$ と $U_i$(および両者が用いる公共的変数)の仕様は，相互排除の問題の解を効果的に構成している．

他のどの $S_j$ も終了が阻止されているとき，かつそのときにのみ，タスク $S_i$ が終了する(したがって $T_i$ が開始可能となる)ように $S_i$ を設計しよう．$U_i$ の主な機能は，$T_i$ が $R$ を使い終ったことを他のタスクに知らせることである．したがって，公共的変数を操作することにより全ての $j$ に対し $\cdots \underline{S}_i \cdots \underline{S}_j \cdots \bar{U}_i \cdots$ の形の部分系列が生じ得ないように，タスク $S_i$ は実行系列中で $\underline{S}_i$ に続く事象を制御することになる．さらに，$R$ の使用要求の待ち行列に対する機構もまた，$S_i$ および $U_i$ に含まれるであろう．後で引用するために，システム $C$ において $R$ を使う各 $T_i$ に対し，タスク $S_i$ と $U_i$ を追加して拡張したシステムを $C'$ と書くことにしよう．

$S_i$ と $U_i$ の仕様を図 2.4-2 に示す．ここで公共的変数は $(n+1)$ 要素の整数型配列 $Q[0], \cdots, Q[n]$ および整数型変数 $p$ である．配列 $Q$ の目的は，$R$ を使おうとして待っているタスクを FIFO 順のリストとして蓄えることである．これには，待ち行列の先頭タスクの番号を $Q[0]$ に記入し，もしも $S_i$ が既に待

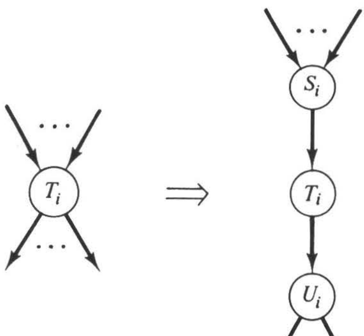



図 2.4-1　補助タスク $S_i$ と $U_i$ の導入

$$
\begin{aligned}
S_i:&\quad \begin{cases} i1: & Q[p] \leftarrow i;\ p \leftarrow i; \\ i2: & \textbf{if } Q[0] \neq i \textbf{ then go to } i2; \end{cases} \\
U_i:&\quad\ \ i3:\ \ \textbf{if } p \neq i \textbf{ then } Q[0] \leftarrow Q[i] \\
&\qquad\qquad\qquad \textbf{else}\{p \leftarrow 0;\ Q[0] \leftarrow 0\}
\end{aligned}
$$

**図2.4-2**　補助タスクのプログラミング

っているならば待ち行列内の $S_i$ の次のタスクの番号を $Q[i]$に記入することによって行われる．このように，待ち行列は単純な連結リストとして $Q$ に記憶される．整数 $p$ は待ち行列の末尾のタスクの番号，すなわち最後に待ち行列に入ることになったタスクの番号である．条件 $p=0$ かつ $Q[0]=0$ は空行列を意味するのであるが，これを初期条件にしよう．

タスク $S_i$ は，$R$ を使うという $T_i$ の意向を知らせて $T_i$ を待ち行列 $Q$ に入れ，その後 $T_i$ が待ち行列の先頭になるまで段 $i2$ で待つ．タスク $U_i$ は $T_i$ を待ち行列から取除き，待ち行列の先頭には次に実行すべきタスクを置く．もし $T_i$ が待ち行列内の最後のタスクであるならば($U_i$ の実行時に $p=i$)，待ち行列を空にするために変数 $p$ と $Q[0]$の値を0にする．

以上の説明では，同時に実行しているタスク $S_i$ と $S_j$，または $S_i$ と $U_j$ 間で起り得る干渉が考慮されていない．非干渉性は，$S_i$ と $U_i$ 内のいくつかの段が実際には不可分であると仮定することによって保証される．しかし，最初にこれらの仮定を必要とする問題の性質を考察しよう．

$p=Q[0]=0$ であるとき，$S_j$ が実行を開始すると仮定しよう．$S_j$ が段 $j1$ において $Q[p]\leftarrow j$ を実行した後ではあるが $p=j$ と置く前に，タスク $S_k$ がやって来て，段 $k1$ の代入文を両方共(高速に)実行することに成功したと仮定する．明らかに，$Q[0]$は $S_k$ によって書き直されるから，$T_k$ は直ちに実行できるけれども $T_j$ の要求は“失われ”てしまう．一時に一つの $S_i$ のみが段 $i1$ を実行し得るように段 $i1$ が不可分であると仮定するならば，この干渉の可能性は除去される．そのような操作を一つの**基本操作文**(primitive)，**enqueue**$(i)$ ($i$ はこの基本操作を呼び出したタスク $S_i$ から与えられるタスク番号のパラメータである)という形で表現することは自然であろう．“基本操作文”という語は，対応する操作が，タスクのプログラム作成者には到達不可能なより低いレベルで実現されるという事実に対する注意を喚起している(しかし，一般に，基本操作は上記のように明記しない限り不可分であると仮定することはできない)．

不可分な基本操作**enqueue**($i$)が採用されると仮定しても，タスク $U_i$ から生ずる問題がまだある．たとえば，段 $i3$ で $p$ が $U_i$ によって調べられて $i$ に等しいことが分ったと仮定する．$p$ と $Q[0]$が $U_i$ によって0にされる前に $S_k$ がやって来て，基本操作 **enqueue**($k$)を実行する．しかし，段 $k2$ が実行される前に, $U_i$ が $p$ と $Q[0]$ を0にすることに成功する．明らかに $T_k$ は決して実行されないことになる．前のように考えると，段 $i3$ が不可分な基本操作文，たとえば **remove**($i$)から成るとし，任意の $j$ に対する基本操作 **enqueue**($j$)とは相互排除されると仮定することにより干渉を除去することができる．実際，段 $i1$ と $i3$ におけるこれらの基本操作で，相互排除問題に対する安全な解が得られることは容易に確かめられる†.

資源の利用効率についてここで二，三の注解を加えるのが適当であろう．これまで説明してきた手続きでは，待機中のタスク($S_i$)がプロセッサを占有してしまうので，比較的多くのタスクが $R$ を競合するシステムにおいてはプロセッサの利用効率を高くできない原因となり得る．これは“(プロセッサ)忙殺型の待ち(busy form of waiting)”と呼ばれ，プロセッサが立往生するだけでなく，共通の変数に繰り返し到達しようとすることから記憶装置での干渉も生ずる．中央の管理に従わないいくつかの自律的な計算機から成るシステムでは，段 $i2$ におけるループを個々の計算機の割込みシステムに組入れてしまうことが望ましく，これにより，$R$ を待つ間これらの計算機は，CPU を生産的な利用に振り向けることができる．

多重プロセッサ・システムにとって，“忙殺型の待ち”は，単一のオペレーティング・システムによる資源の集中管理論に与する基本的論拠のほんの一例となるに過ぎない．そのように作成されたオペレーティング・システムの下では，タスク $S_i$ および $U_i$ はスーパバイザ呼出し以上のものではないことになる．そこでは，プロセッサの再割当ておよび資源の一般的スケジューリング機能と共に，図2.4-2の機能がオペレーティング・システムによって遂行される．

### 2.4.3 一般問題の解

同期の問題を扱う次節では，**enqueue**($i$)および **remove**($i$)の機能を果すようにも利用できる別の基本操作を導入する．その基本操作の設計は，FIFO

† プログラムの性質(たとえば $S_i$ や $U_i$ の性質)に関する我々の議論は，本節と次節では，大部分が非形式的である．これらの議論を形式化する手法も開発できるが [14, 15]，より厳密な接近法を除外することは，必要となる余分な機構(を考えないですむこと)や，非形式的な議論の方が説得力を持つことから，十分に理由のあることである．

規則とは違う優先権順序付けを実現するようにも修正できるということを注意しておく．しかし，本節の残りの部分は，資源の集中管理が行えず，かつ公共的変数に対する不可分操作が記憶に対する1回の読取りまたは1回の書込みだけであるときの相互排除の実現に当てられている．

問題の解を前と同様に表現し，相互排除を実現するためにタスク $S_i$ と $U_i$ を設計しよう．$S_i$ は，他のどの活動中の $S_j$ も終了が阻止されているとき，かつそのときのみ終了する．設計が難しいのは，主として，無定限に阻止されることがないことを保証する安全な解を求めようとするからである．

整数型配列 $c[1:n]$（管理用）と整数型変数 $r$ から成る公共的変数を定義する．最初は全ての $i$ に対して $c[i]=0$ である．タスク $S_i$ と $U_i$ の設計を図2.4-3に示す．ベクトル $c$ は，$S_i$ が活動中であるときに限り $i$ 番目の要素が1になる．タスク $T_i$ が $R$ を使っているならば，$r=i$ かつ $c[i]=2$ である．$R$ を使っている $T_i$ がなければ，$R$ を使った最後のタスクを $T_j$ としたとき $r=j+1$（$j=n$ の場合は1）である．

図2.4-3のアルゴリズムが安全な解を与えるということの証明では，余分な注釈を加えずに，$S_i$ と $U_i$ が望み通りの仕方で協同作業（通信）をすることを示すことに専念しよう．この証明に続いて，安全な解の作成に潜む微妙な点を十分明らかに示す例を詳細に述べる．この例は，必ずしも証明を追うのに必要ではないけれども，証明の後半を吟味する間に研究すれば理解の手助けになろう．その論法は，Dijkstra [16] や Knuth [13] のそれと同列であり，次の形で結果を示すのが適当と思われる．

```
      ┌ 1. c[i] ← 1
      │
      │ 2. for j = r step 1 until n, 1 step 1 until n do
      │      if j = i then go to 3 else if c[j] ≠ 0 then go to 2
      │
      │ 3. c[i] ← 2
S_i  ─┤
      │ 4. for j = 1 step 1 until n do
      │     if (j ≠ i) and (c[j] = 2) then go to 1
      │
      └ 5. r ← i
T_i     6. { Rの使用 }
U_i   ┌ 7. r ← if i = n then 1 else i + 1
      └ 8. c[i] ← 0
```

**図2.4-3**　$T_i$ の補助タスクのプログラミング

**定理 2.4**

一時には高々一つのタスクによってしか使えない資源 $R$ を必要とするタスク・システムを考える．タスク $S_i$ と $U_i$ は上述のように導入されていると仮定する．$S_i$ と $U_i$ に対して図 2.4-3 で定義したプログラムは相互排除問題に対する安全な解を構成している．

**証明** 与えられたシステム $C$ において，$T_1, \cdots, T_n$ が $R$ を必要とする独立なタスクであるとする．最初に証明するべきことは，タスク $T_1, \cdots, T_n$ のうち一時には高々一つしか活動中になり得ないということである．このことは，タスク $S_i$ が段 4 で配列 $c$ の他の要素を調べる前の段 3 で，$c[i] \leftarrow 2$ を実行することから直接得られる．任意の $j \neq i$ に対して $S_j$ による段 3 $(c[j] \leftarrow 2)$ の実行と同時に，$S_i$ がもしも段 5 に到達したとすれば，$S_i$ は $S_j$ より前に段 3 を完了したことは確かなはずである．したがって，$S_j$ は $c[i] = 2$ に出会って段 1 に戻ることになる．結局，一時には高々一つのタスク $T_i$ しか活動中になり得ない．

上記は，二つ以上のタスクが段 3 と段 4 内を並行して実行するようなことが実際に起っても構わないことを意味している．このような事態が生じ得ることを確かめるため，$S_i$ が段 2 の実行を開始したと仮定しよう．与えられた $r$ の値に対し，第 1 番目の位置を $r$ とする $(1, 2, \cdots, n)$ の巡回置換を $(r, r+1, \cdots, r-1)$ で表す．段 2 で $S_i$ は，$(r, r+1, \cdots, r-1)$ の順序に従って $c[r]$，$c[r+1]$，$\cdots$，$c[i-1]$ を調べる．これらの値がそれぞれ 0 であるとき，しかもそのときにのみ $S_i$ は段 3 に到達する．したがって，順序付け $(r, r+1, \cdots, r-1)$ では $k$ が $i$ より前にあるけれども，$S_i$ が $c[k] = 0$ であることを見てしまうまで $S_k$ が段 1 を完了しないならば，$S_k$ は $S_i$ と一緒に段 3 に到達し得る．

安全性を証明するため，どのタスク一つをとっても段 1～4 を無定限に循環するものはないということを示さなければならない．まず，二つ以上のタスクから成る集合が無定限に循環するようなことはないことを示そう．$S_{i_1}, \cdots, S_{i_k}$ $(k \geq 2)$ が段 1～段 4 内で循環していると仮定する．一般性を失うことなく，順序付け $(r, r+1, \cdots, r-1)$ では，$i_1$ は $i_2$ より前にあり，$\cdots$，$i_{k-1}$ は $i_k$ より前にあると仮定してよい，各 $S_{i_j}$ はある有限時間内に段 2 を実行するはずである．段 2 の操作に従って，$S_{i_j}$ は $(r, r+1, \cdots, r-1)$ の順に $c$ の値を調べる．その結果，当然，$S_{i_1}$ 以外のあらゆるタスクは段 2 でその進行が停滞しなければならなくなる．$S_{i_j}$ 以外の各タスクの $c$ の値は 0 であるから，$S_{i_1}$ はいずれは段 5 に到達し，続いて $T_{i_1}$ が開始する．結局，二つ以上のタスクから成る集合が

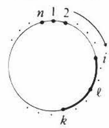
無定限に $R$ を使うことを阻止されるようなことは起り得ない．このことは，どの特定のタスクも $R$ の使用を無定限に阻止されることはないということを示すのに使われる．

次に，特定のタスク $S_i$ が $R$ の使用を無定限に阻止されると仮定しよう．二つ以上のタスクから成る集合が無定限に阻止されることはないから，他のタスク，たとえば $S_k$ が存在して無定限に繰返して($T_k$ を)開始しなければならない．さて，$c[i] \neq 0$ であるのに，$S_k$ が随意に段2を通過できるのは，$r$ があるタスクによって，$(r, r+1, \cdots, r-1)$ という順序付けでは $k$ が $i$ より前になるような値を与えられているときに限られる．このような値を $r$ に与えるタスクは，$S_k$ とは違うものでなければならない．なぜならば，そうでないとすると，$S_k$ は† 段7を訪れたときに $r$ に $k+1$($k=n$ ならば1)を代入するので，次の循環の段2では $i$ が $k$ より前になるような順序で探索を始めてしまうからである．以上のことは，($S_k$ が† 段7を実行した後の順序付け)$(r, r+1, \cdots, r-1)$ において $l$ が $k$ の前かつ $i$ の後になるような††，無定限に繰返して($T_l$ を)開始するタスク $S_l$ が存在しなければならないことを示している．しかし，もともと $k$ はこの(無定限に繰返して開始するという)性質を持つ $i$ の最初の後続者であると仮定しておけば，(上記のような $l$ の存在は)矛盾を生ずる．■

前述の議論を使って，相互排除問題で起り得る競合条件についてのより深い洞察が次の一般的な例から得られるが，この例は，不利な境遇ではタスクの待ち時間がいかに長いものであり得るかを示している．説明を簡単にするため，制御が図2.4-3の段5から段8まで通過するときはいつも，タスクは($R$ を使って)**一働きする**ということにしよう．また，タスクが図2.4-3の段 $i$ を丁度完了したとき，そのタスクは**位置 $i$ にある**ということにしよう．あらゆる時間間隔を通じて，タスクが実行する相対的な速度を完全に自由に選択できるものと仮定する．

$R$ を競い合っている $n$ 個のタスク $T_1, \cdots, T_n$ があるとし，我々の例に対する初期条件は $r=n$ で，$S_1$ は位置1にあり，タスク $S_2, \cdots, S_n$ は全て位置2

† 訳注　“$S_k$ に対応する $U_k$” の意．
†† 訳注　右図の太線の区間内に $l$ があることを言っているのである．



にあると仮定する†. 次のような動作の系列を選択する.

1. $S_n$ が位置5に進んだ後, (非常に素早く)$T_n$ が一働きする($r = 1$, $c[n] = 0$ になる). この時点では, $r = 1$ で $c[r] = 1$ だから, $S_n$ は(次の)段2を通過できない. そこで, $S_1$ が $r$ の現在の値を見る前に,
2. $S_{n-1}$ が位置5に進み $T_{n-1}$ が一働きする($r = n$, $c[n-1] = 0$ になる)ものとする. 今や $S_n$ は段2を通過することができるので, 選ばれる動作は次の通りである.
3. $S_n$ が位置5に進み, $T_n$ がもう1回一働きする($r = 1$, $c[n] = 0$ になる). この時点で $S_{n-1}$ と $S_n$ は段2を通り抜けることはできない. $S_1$ が $r$ の現在の値を見る前に,
4. $S_{n-2}$ が位置5に進み, $T_{n-2}$ が一働きする($c[n-2] = 0$, $r = n-1$ になる). 段2の論理により, $c[n-1] = c[n] = 0$ であるから, 今や次のように仮定することができる.
5. $S_n$ と $S_{n-1}$ がこの順に位置2まで進む††. これで上述の動作1～3を繰返す位置に来たことになるので,
6. 上記の1～3に従って $T_n$ がさらに2回, $T_{n-1}$ も1回一働きする($r = 1$, $c[n-1] = c[n] = 0$ になる). 引続き $T_1$ に $R$ を使わせないためには, (上記の4と同様に)$S_1$ が $r$ の現在の値を見られないまま次の動作に入ると仮定しなければならない.
7. $S_{n-3}$ が位置5まで進行し $T_{n-3}$ が一働きする($r = n-2$, $c[n-3] = 0$ になる).

これで枠組がはっきりしたであろう. この時点で $S_{n-2}$, $S_{n-1}$ および $S_n$ が†††位置2まで進んで上記の1～6が繰返され, $T_n$ はあと4回, $T_{n-1}$ は2回, $T_{n-2}$ は1回一働きする. 究極的には,初期のタスクの集まり††††が尽きて $r = 1$ となるから, 上記の4や7と同様に, $T_1$ が待っていた一働きをするまでは, $T_2, \cdots, T_n$ が次の一働きをすることができない(すなわち, $S_2, \cdots, S_n$ が段2を通過できない)ような時点に到達する. この時点に至るまでに $T_1$ は, $T_2$ が1回, $T_3$ が2回, $T_4$ が4回, …, $T_n$ が $2^{n-2}$ 回($n \geq 2$)一働きするのを待った

† 訳注 位置2にあり, 段3を開始しようとしている状態である. このような状態があり得ることは, 定理2.4の証明や下記の動作5を吟味することによって, 容易に確かめられるであろう.

†† 訳注 段3を開始しようとしている状態である.

††† 訳注 この逆の順序で

†††† 訳注 最初, 段2にあったタスクの集まりのこと.

ことになる．特に，$T_n$ は最初から始めて1回おきに，$T_{n-1}$ は2回目から始めて4回目毎に，$T_{n-2}$ は4回目から始めて8回目毎にという具合に，それぞれ一働きしたことになる．このように，$T_1$ は最初の一働きをするまでに併せて $\sum_{i=2}^{n} 2^{i-2} = 2^{n-1} - 1$ 回だけ待たなければならない．たとえば，系列 $T_5T_4T_5T_3T_5T_4T_5T_2T_5T_4T_5T_3T_5T_4T_5$ は，$n=5$ の場合に $T_1$ が待たなければならない全ての一働きとその相対的な順序を示している†.

## 2.5 同　期

いくつかのタスクがタイミングを取り合って正しく順序付けられるように実行したいという要請の実現は，**タスクの同期**(task synchronization)と呼ばれている．この概念はタスク・システム間の同期を考えるときと全く同じである．大まかにいえば，同期の問題はタイミングと本章で既に扱った順序付けの問題を含んでいる．しかし，本節で後に示すように，ある循環的なシステムにおけるタスクの終了の回数が，並行して実行している他の循環的なシステムのタスクの終了回数に依存するようにしたいといった状況を同期問題の制約内に含むことが望ましい．

同期の条件は，タスク・システムの先行グラフの中に明確に表現される．2.2節で扱った決定性問題は，先行関係，したがって同期の制約がなぜ課せられなければならないかという理由の一つに関係しているし，閉鎖は資源に対する競合によって生ずるから，その予防法もタスク間に先行制約を課すことによって責任を果すことができる．相互排除もまた同期の問題の一つに該当するけれども，先行関係に方向性を必要としないという点で，すなわち，相互排除されるタスクは，時間的に(重ならないように)順々に実行されなければならないが，きちんとした順序まで規定する必要はないという点で異なっている．

本節の目標は，同期をとる必要のあるタスクがそれ自身のプログラムの中でそれを実現できるような，いわゆる**同期基本操作**(synchronization primitive)をいくつか示すと共に，オペレーティング・システムの設計において生ずるいわゆる**生産者消費者**(producer-consumer)同期問題でそれらがどのように使われるかを議論することである．前節の基本操作 **enqueue** と **remove** のように，これらの基本操作も通信用の公共的変数に作用する．

タスクは互いに公共的変数を介して通信を行うけれども，それらは**信号器**(semaphore)として知られる単なる整数型の変数から成っている．二つの基

† いくぶん手の込んだアルゴリズムになるという代償を払えば，任意のタスクの最悪の場合の待ちを減らすことができ[27, 28]，その最小のものは $n$ に関して線形である．

本操作文 **wait**$(s)$ および **send**$(s)$ が，信号器 $s$ を操作するために定義される†．これらの操作の定義を図 2.5-1 に示す．信号器は，タスク・システム生成時に初期化され，その値は実現しようとする同期の型によって決められる．

前節と同様に，基本操作の実行に関していくつかの仮定を置くことができる．これらの基本操作は，(少なくとも部分的には)ハードウェアに組込むことができるであろうし，タスクによって呼出されるオペレーティング・システム中のルーチンによって実行することもできるであろう．また，タスク自身によって実行することもできよう．しかし，いずれの場合にもある一連の操作は不可分であることが保証されなければならない．特に，記憶に対する 1 回の読取りおよび 1 回の書込みだけが不可分操作であるという仮定の下では，ある信号器 $s$ を操作する際に二つのタスクが互いに干渉し合う可能性がある．たとえば，二つのタスクが $s \leftarrow s+1$ を実行しつつあると仮定しよう．一般に，どちらかのタスクが増加した値を $s$ に書込む前に，両タスクとも $s$ の古い値を読んでしまうということが起り得るであろう．その場合には，両者の実行完了後，$s$ は $s+2$ になるのではなく，$s+1$ になってしまう．したがって，**wait**$(s)$ および **send**$(s)$ における代入操作 $s \leftarrow s-1$ および $s \leftarrow s+1$ は，それぞれ不可分であることが必要である．

これらの操作の簡単な適用例として，与えられたタスク・システム $C$ の先行関係を実現する問題がある．このために，$C$ のグラフ $G$ 内の発端タスク以外の各タスク $T_j$ に対して信号器 $s_j$ を定義しよう．各タスク $T_i$ に対するプログラムが与えられたとき，次のようにして新しいタスク $T'_i$ を作る．すなわち，各非発端タスク $T_j$ の前に基本操作文 **wait**$(s_j)$ を置き，各非終端タスク $T_j$ の後にはその直接の後続者 $T_k$ の各々に対する基本操作文 **send**$(s_k)$ をそれぞれ置く．$s_j$ の初期値は，$T_j$ の直接の先行者の数が $d_j$ であるならば，$-(d_j-1)$

```
wait(s):      s ← s − 1
         L:   if s < 0 then go to L

send(s):      s ← s + 1
```

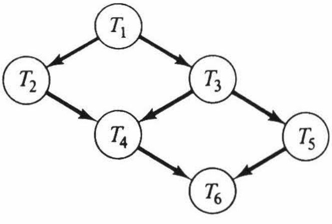
図 2.5-1　同期基本操作

† これらの操作は，当初，Dijkstra[4]によって，$P$ および $V$ 操作と呼ばれていた．これらの記号の原語は，オランダ語であって英語ではない．

に設定される．例を図2.5-2に示す．**wait**($s_j$)が(図2.5-1の)段 $L$ を実行中であるとき，$s_j$ の絶対値は $T_j$ の未完了の(直接の)先行者数を表すことに注意しよう．これらの先行者は，$T_j$ が実行を開始できるようになる前に基本操作**send**($s_j$)をそれぞれ実行しなければならない．(問題2.15参照).

忙殺型の待ちのために生ずる資源の利用率低下の問題は，前節で議論したタスク $S_i$ と同様に **wait** 基本操作にも当てはまるが，その解決法も同じである．特に，**wait** 基本操作の待ちのループを割込みシステム内で実現することもできるし，信号器変数を介するタスク間通信を調整する集中化されたシステム・ルーチンを持つようにしてもよい．

同期のもう一つの重要な適用例は，生産者と消費者という関係を持って実行を行う循環タスク・システムにおいて生ずる．たとえば，システム $C_1$ は出力コマンドを生産してそれを $N$ 個のセルから成る緩衝記憶に置き，これに並行に $C_2$ はその緩衝記憶から出力コマンドを取出して実行する．明らかに，$C_1$ が一杯になった緩衝記憶にコマンドを置くことを，また $C_2$ が空の緩衝記憶からコマンドを取出すことを阻止しなければならない．より抽象的にいえば $C_2$ が完了する回数は $C_1$ のそれを越えてはならないし，しかも $N$ を越える程 $C_1$ のそれより少なくなってはならない．これらの性質は実行系列を用いて容易に定式化される．




$T_1'$: $T_1$, **send**($s_2$), **send**($s_3$)

$T_2'$: **wait**($s_2$), $T_2$, **send**($s_4$)

$T_3'$: **wait**($s_3$), $T_3$, **send**($s_4$), **send** ($s_5$)

$T_4'$: **wait** ($s_4$), $T_4$, **send** ($s_6$)

$T_5'$: **wait** ($s_5$), $T_5$, **send** ($s_6$)

$T_6'$: **wait** ($s_6$), $T_6$

**図2.5-2**　先行関係の制約の実現

**定義** 2.6

$C = C_1{}^m \parallel C_2{}^n (m \geq 1, n \geq 1)$と定義し，$\alpha$を$C$の実行系列とする．$\alpha$の任意の語頭$\beta$内の$C_i$の終端タスクの終了事象の数を$N_i(\beta)$で表す．与えられた$k$に対し，$\alpha$の全ての語頭$\beta$について$N_1(\beta) = 0$であるか$N_1(\beta) \leq N_2(\beta) + k$であるならば，$C_1$は$\alpha$において$C_2$に$k$-同期しているという．

先程の例でこの定義を説明するため，大きさ$N$の出力緩衝記憶が最初は空であると仮定する．このとき，$C_1$が一杯の緩衝記憶に書込もうとしないためには，あらゆる実行系列で$C_1$は$C_2$に$N$-同期していなければならない．同様に，$C_2$が空の緩衝記憶から読取ろうとしないためには，$C_2$は$C_1$に0-同期していなければならない．これは**相互**(mutual)同期の一つの形であり，この性質については後で考察する．

定義2.6の非対称な性質と，$m > n + k$ならば望む性質を持った$\alpha$が存在しないという事実にすぐに気がつく．また，与えられた$\alpha$に対し，$C_1$が$C_2$に$k$-同期していることと，$C_2$が$C_1$に$(-k)$-同期していることとは同じではないことにも気がつく．特に，$k$が非負ならば，$C_1$の$\alpha$における$C_2$への$k$-同期は，$\alpha$の全ての語頭にわたって$C_1$の完了の回数が$C_2$のそれを決して$(k+1)$回以上，上回ることはないということを意味している．一方，$k$が負ならば，$\alpha$における$C_2$の$C_1$への$k$-同期は，$\alpha$の全ての語頭にわたって$C_2$の完了の回数が0であるか$C_1$のそれよりも少なくとも$(-k)$回は少ないかのいずれかである．すなわち，$C_2$の実行が始まってしまったならば，その完了は少なくとも$(-k)$回分は$C_1$のそれよりも遅れなければならない．いずれ分るように，本節で議論される適用例は，非負のパラメータのみの同期に関連する問題である．

目下の問題は，二つのタスク・システム内に正しく組込まれるとき，適切な同期の特性が得られることを保証する基本操作を考えることである．これを用いて新たに定義されるタスク・システムの並列結合に対し，要求された同期の特性が全ての正当な実行系列に対して成立することが望ましい．これは，**wait**および**send**の基本操作を用いて達成することができる．

$C_1$および$C_2$はいずれも閉じたタスク・システムであると仮定し，$x$を信号器変数とする．新しい(閉じた)システム$C'_1 = [\mathbf{wait}(x);\ C_1]$は唯一つの操作**wait**$(x)$から成る新しい発端タスクを$C_1$の前に付加したシステムであると定義する．同様に，$C'_2 = [C_2;\ \mathbf{send}(x)]$は，新しい終端タスク**send**$(x)$を$C_2$の後に付加した閉じたシステムを表すものとする．

次に，$x$の初期値は$k$であるとして，$C = (C'_1)^m \parallel (C'_2)^n$の正当な実行系列

を考えよう．明らかに，$C'_1$ の循環は **wait**$(x)$操作によって検査された時点で $x \geq 0$のときにのみ開始することができる．**send**$(x)$が $x$ を1だけ増やし **wait** $(x)$が$x$ を1だけ減らすから，$C$ の任意の正当な部分実行系列では，完了した $C'_1$ の循環の数は $C'_2$ のそれを $k \geq 0$ の場合は $k$ を越えて上回ることはなく，$k < 0$ の場合は少なくとも $-k$ だけは少なくなければならない．この性質をまとめると次のようになる．

**命題 2.7**

$C_1, C_2, C'_1$ および $C'_2$ は上で定義されたものとし，$k$ を信号器 $x$ の初期値とする．$C = (C'_1)^m \| (C'_2)^n$(ただし，$m \geq 1$, $n \geq 1$)と定義し，$C$ の正当な実行系列の集合を $E$ で表す．このとき，あらゆる $\alpha \in E$ において，$C'_1$ は $C'_2$ に $k$-同期している．さらに，$m \leq n + k$ であるときかつそのときに限り，$E$ は空でない．

さて，相互同期の概念に戻ろう．

**定義 2.7**

$C = C^m{}_1 \| C^n{}_2 (m \geq 1,\ n \geq 1)$とし，$\alpha$ を $C$ の実行系列とする．$\alpha$ において $C_1$ が $C_2$ に $k$-同期しており，かつ，$C_2$ が $C_1$ に $l$-同期しているならば，$C_1$ と $C_2$ は $\alpha$ で$(k, l)$-同期しているという．

上の性質を持った実行系列が存在するためには，整数 $k, l, m$ および $n$ はある制約を満たさなければならない．それは次の通りである．

**命題 2.8**

$C_1, C_2$ および $C$ は定義2.7 におけるように与えられたとする．$C_1$ が $C_2$ に $k$-同期しており，$C_2$ が $C_1$ に $l$-同期しているような実行系列の集合を $E$ とする．$E_1$ が空でないのは，次の制約を満たすとき，しかもそのときに限る．

(2.5.1)　　$$k + l \geq 1 \text{ かつ } -l \leq m - n \leq k$$

この結果を確かめるため，$\beta$ が $E$ に属するある実行系列 $\alpha$ の語頭であると仮定する．このとき，

(2.5.2)　　$$N_1(\beta) \leq N_2(\beta) + k$$

かつ

(2.5.3)　　$$N_2(\beta) \leq N_1(\beta) + l$$

が成り立つ．これらの不等式を加えると，$k+l \geq 0$ が得られる．後は，$k+l=0$ が成り立ち得ないことを示すことが残っている．ところが，$k=-l$ であると仮定すると，**全ての**語頭 $\beta$ が $N_1(\beta)=N_2(\beta)+k$ を満たさなければならないことになり，これは明らかに不可能なことである．したがって，$k+l \geq 1$. 最後に，式(2.5.2)および式(2.5.3)において $\beta=\alpha$ と選ぶと，$-l \leq m-n \leq k$. 命題の逆の部分は，適切な実行系列を構成する簡単なアルゴリズムを与えることによって示すことができる．その詳細は省略する．

特別な場合である $(k=0, l=1)$ および $(k=1, l=0)$ は，$C_1$ と $C_2$ の完了が交互に生じ，$(k=1, l=0)$ の場合は最初の完了は $C_1$ の方であり，$(k=0, l=1)$ の場合は $C_2$ の方であるという要件に対応していることに注意しよう．ここで，相互同期を実現するときの **wait** および **send** 操作の使い方を述べよう．命題2.7の対称的な適用から，直ちに次の結果を得る．

**命題 2.9**

$C'_i$ が命題2.7におけるように与えられているとし，$C''_1=[C'_1;\ \mathbf{send}(y)]$ かつ $C''_2=[\mathbf{wait}(y);\ C'_2]$ と定義する．$C=(C''_1)^m \parallel (C''_2)^n$ の実行系列の集合を $E$ とする．$k, l$ をそれぞれ信号器 $x, y$ の初期値とする．このとき，全ての $\alpha \in E$ において，$C''_1$ と $C''_2$ とは $(k, l)$ 同期している．

図2.5-3は，$C_1$ と $C_2$ の相互同期で使われる **wait** 文と **send** 文の配置を示している．

さて，前の例で示した生産者消費者問題をもう少し詳しく考えてみよう．相互同期のこの重要な応用は，一般にタスク間通信のためのモデルであり，特に入出力緩衝記憶のためのモデルである．置き場所が $N$ 個ある緩衝装置を $B$ とする．生産者と呼ばれるタスク(すなわち，単一のタスクから成るシステム)は

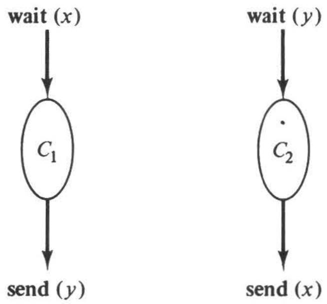



**図 2.5-3**　相互同期したタスク・システム

緩衝装置に生産品を置き，消費者と呼ばれるタスクがそれを持去る．生産品は置いた順に持去られる． これらの二つのタスクに対するプログラミングを図2.5-4に示す．$in$ および $out$ を整数型変数，その初期値を $in = out = 1$ として，$B$ 上で定義される次の二つの操作がある．

$$input(w)\colon\ B[in] \leftarrow w;\ in \leftarrow \textbf{if}\ in = N\ \textbf{then}\ 1\ \textbf{else}\ in + 1$$
$$output(w)\colon\ w \leftarrow B[out]\ ;\ out \leftarrow \textbf{if}\ out = N\ \textbf{then}\ 1\ \textbf{else}\ out + 1$$

信号器 $x$ は初期値0であり，任意の時点におけるその値は緩衝装置内で塞っている置き場所の数を表している． 信号器 $y$ は初期値 $N$ であり，任意の時点におけるその値は緩衝装置内の空いている置き場所の数を表している． 命題2.9は，生産者は一杯になった緩衝装置内に格納しようとすることを阻止され，消費者は空の緩衝装置から取出そうとすることを阻止されていると結論するのに使うことができる．この二つのタスクの正しさの証明を完結するためには，生産者と消費者が緩衝装置内の同じ置き場所に同時的に到達しようとすることはないということを示す必要がある． これが起るのは， $in = out$ のときだけであり，条件 $in = out$ が成立するのは，$x$ の値が0であるか $y$ の値が0であるとき，しかもそのときのみであり，いずれの場合でもどちらか一方のタスクがその進行を阻止されている．

三つ以上のタスク・システムが関係する同期への一般化を考えることは難しくない． たとえば， $n(n-1)$ 個のパラメータ(各タスク対に一個)を用いて $n$ 個のタスクの相互同期に一般化することができる．しかしながら，そのような一般化は，主に理論上の興味の対象であるように見える(問題2.2参照)． 実用上の興味からの一般化の一例は，複数の生産者あるいは複数の消費者がいる場合の複数生産者消費者問題である．この場合の望ましい同期の特性は，非形式的にいえば次のようになる．すなわち，生産者タスクの完了の回数の和は，消費者タスクの対応する和を $(N+1)$ 回以上は上回ることができない． さらに，前者の和は後者の和よりも小さくなってはいけない．

この複数生産者消費者問題の解を実現するためには，待ち行列の機構を含む

| | | | |
|---|---|---|---|
| 生産者： | **produce** (*w*) | 消費者： | **wait** (*x*) |
| | **wait** (*y*) | | **output** (*w*) |
| | **input** (*w*) | | **send** (*y*) |
| | **send** (*x*) | | **consume** (*w*) |
| | **goto** producer | | **goto** consumer |

図2.5-4　生産者消費者問題

```
enter (i, Q):      i1:  Q[p] ← i; p ← i;
                   i2:  if Q[0] ≠ i then goto i2;

remove (j, Q):     if p ≠ j then Q[0] ← Q[j]
                            else {p ← 0; Q[0] ← 0} ;
```

**図 2.5-5　enter および remove の定義**

**enqueue** および **remove** のような基本操作を使わなければならない．これらの基本操作を上の問題に適用するには，追加のパラメータとして公共的変数(配列) $Q$ を導入することだけが必要である．しかし，**wait** 基本操作と両立するように，図 2.4-2 の段 $i2$ 内の待ちのループと段 $i1$ の **enqueue** 基本操作とを結合する．図 2.5-5 に示すように，新しい基本操作 **enter**($i$, $Q$)を段 $i1$ および $i2$ から成るものと定義し，**remove** 基本操作は $Q$ をパラメータとして含むものに拡張しよう．これらの基本操作を用いて，$i$ 番目の生産者と $j$ 番目の消費者のプログラムを図 2.5-6 のように作る．一般に，タスクはまず緩衝装置へ到達するために待ち，次いで生産者であれば空の置き場所が，消費者であれば空でない置き場所ができるのをそれぞれ待たなければならない．

図 2.5-6 の基本操作文で示された操作がタスク自身の側で遂行される場合は，適当なハードウェア，オペレーティング・システムの利用法，利用可能性などに依存するということを再度強調しておく．しかし，公共的変数への操作が不可分であるということは引続き仮定しておかなければならない．なぜかというと，もしそうでなければ，前節の一般の相互排除問題に対して与えたのと同様のアルゴリズムが必要になるからである．

同期問題の議論は，基本操作文 **wait** および **send** に限ってきた．このほかに，同期をとるための基本操作文の組がいくつも定義され研究されてきている [17, 18, 19]．問題 2.13 ではいくつかの例を考える．

```
生産者 i:  produce (w)              消費者 j:  enter (j, Q2)
           enter (i, Q1)                       wait (x)
           wait (y)                            output (w)
           input (w)                           send (y)
           send (x)                            remove (j, Q2)
           remove (i, Q1)                      consume (w)
           goto producer i                     goto consumer j
```

**図 2.5-6　複数生産者消費者問題**

## 演 習 問 題

**2.1** 教科書[6]でも仮定されているように，メッセージ，カード・イメージやデータといった“資源”が，再使用可能ではなく**消費性**(consumable)である“通信閉鎖(communication deadlock)”の問題を考えよう．消費性の資源を生産するか，それを使う(消費する)か，あるいはその両方を行うかのいずれかであるようなタスクの集合を想定する．当然，資源を生産するタスクから成る部分集合は空であってはならない．任意に与えられた資源について，生産できる量には制限がないと仮定する．すなわち，**生産者タスクが実行可能である**(同時に他のタスクからの資源を待っていない)と仮定すると，望むだけの資源を生産できることになる．タスク間の相互作用は今や明白であり，各タスクによって必要とされ他のあらゆるタスクから供給される資源によってその細部が指定される．問題は再使用可能な資源のシステムとの本質的な違いを確認することに注意しつつ，そのようなシステムのモデルを作り閉鎖の適当な定義を与えることである．この定義に基づいて閉鎖を検出するアルゴリズムを作り，予防と回避の問題を考察せよ．

**2.2** 独立な循環的タスク $T_1, \cdots, T_n$ から成る任意の集合を考える．各 $i, j$ に対し，$T_j$ が $T_i$ に$k_{ij}$-同期していなければならないような整数 $k_{ij}$ が存在する．これらの制約は次のようにグラフで表現できる．節点はタスク $T_i$ $(1 \leq i \leq n)$ であり，各タスクから他のあらゆるタスクへの適当な同期のパラメータでラベル付けされた枝を持つグラフ(同期グラフと呼ぶ)を定義する．すなわち，$T_i$ から $T_j$ への枝はラベル $k_{ij}$ を伴う．$k_{ij} = \infty$ ということは，$T_j$ が $T_i$ に制約されないということを意味することに注意しよう．

全ての同期の制約を満たす次のような正当な実行系列 $\beta\alpha$ が存在すると仮定する．すなわち，$\alpha$ 内で各タスクが少なくとも1回は実行され，かつ全ての $n \geq 1$ に対して $\beta\alpha^n$ は正当である．このとき，同期グラフは**無矛盾である**(consistent)という．同期グラフが無矛盾であるのは，グラフ内の各閉路の枝のラベルの和が1以上であるとき，かつそのときのみであることを証明せよ．

無矛盾性の問題は，並列プログラム形(parallel program schema)の“停止(termination)”問題[22]やマーク付きグラフ(marked graph)の“生存性(liveness)”問題[23]に類似している．上記の問題を一般化して次のも

のを含むようにできる.

**a.** 信号器に対して任意の整数がそれぞれ減算および加算されるような $P$ および $V$ 操作.

**b.** 信号器が"共有変数"であってもよい場合, すなわち, 二つ以上のタスクが与えられた信号器に $P$ 操作または $V$ 操作を実行してもよい場合.

これらの一般化されたモデルは, Karp と Miller によって研究されたモデルを実現したものであることは容易に分る[22, 24]. 上記の一般化(a)の下での信号器システムに直接基づく議論は[25]に見出すことができる.

**2.3** 次に示す資源の状態に関するもう一つのグラフ表示を閉鎖問題の研究に使うことができる[6]. 状態グラフ $G$ の節点(の集合)は, タスクの連鎖 $C_1$, …, $C_n$ の集合と資源の型 $R_1$, …, $R_m$ の集合の和である. $G$ は, 連鎖 $C_i$ が今要求している各型 $R_j$ の資源単位に対し, $C_i$ から $R_j$ への枝をそれぞれ一本ずつ持つ. また, 型 $R_j$ で今 $C_i$ に割当てられている全ての単位に対し, $R_j$ から $C_i$ への枝がそれぞれ1本ずつある. このように, 型 $R_j$ の現在利用可能な資源数と $R_j$ から出射している枝の数との和は一定で, $w_j$——システム内に存在する型 $R_j$ の総単位数——である. 集合 $\{R_1, \cdots, R_m\}$ 内の節点から(へ)の全ての出(入)射枝は, 互いに素な集合 $\{C_1, \cdots, C_n\}$ 内の節点へ(から)入(出)射しているという意味で, $G$ は**2部**(bipartite)**グラフ**であることに注意しよう.

状態グラフ $G$ は, どの連鎖も現在利用可能な資源で叶えられ得るような資源の要求を持たないとき, (閉鎖の取扱いにとって)**好都合である**(expedient)という. ($C_i$ から $R_j$ への全ての枝は同時に取去らなければならないことに注意しよう. (同じ型に対して)一部の要求だけ満たしておくことはできない). $G$ の節点の部分集合は, それに属するあらゆる節点が少なくとも一つの後続者を持つけれどもどの後続者もその集合の外にはないとき, **結び目**(knot)を構成しているという. 好都合なグラフにおける結び目の存在は閉鎖の十分条件であり, 閉路の存在は必要条件であることを示せ.

各タスクの要求は, 高々1資源単位しか未充足にできないように† システムが制約されているものと仮定する. これらの状況下では, 好都合な状

† **訳注** 一時には1資源単位しか要求できず, 要求が叶えられるまでは次の新たな要求ができないように.

態グラフに結び目が存在していることは閉鎖に対する必要かつ十分条件になっていることを示せ.

**2.4** 任意に与えられた初期状態 $s_0$ に対して $C$ の全ての実行系列 $\alpha, \alpha'$ について $F(M_i, \alpha) = F(M_i, \alpha')\ (1 \leq i \leq m)$ であるとき，システム $C$ は**弱決定性**(weakly determinate)であると定義する．干渉するタスクの存在は，タスク・システムの弱決定性を損うとは限らないことを示せ．3個のタスク $T_1$ と $T_2$ と $T_3$, $T_2 < T_3$ のみから成る半順序，および次の定義域と値域から成る例を考えよ．

$$D_{T_1} = (M_1), \quad R_{T_1} = (M_2), \quad D_{T_2}( = M_2), \quad R_{T_2} = (M_3),$$
$$D_{T_3} = (M_4), \quad R_{T3} = (M_3)$$

**2.5** 2.2節で，タスク $T$ が書いた値は，$T$ が開始した時点の定義域の値に依存していた．もし，状態遷移の定義を修正して，事象 $a_{k+1} = \underline{T}$ によって $R_T$ に書かれる値は状態 $s_k$ で与えられた定義域の値のみに依存するようにしたとする．決定性の問題に関して，この場合でも意味のあるモデルを作り得るだろうか．もしそうならば，基本的な結果に，必要ならばどのような変更が加えられなければならないだろうか.

**2.6** ある資源の同じ写しが $m$ 個あり，各写しは一時には高々一つのタスクにしか使われないものと仮定する．このより一般的なシステムを考慮して，本書に書かれている相互排除問題の解を拡張せよ．もちろん，与えられた資源は効率的に使われるべきであり，資源の写しはその資源を待っているタスクがない場合を除いて，無定限に遊休状態になってはならない.

**2.7** 閉鎖の回避のための MRU モデルの拡張を考えたい．連鎖が要求を行う各時点で，その連鎖の残りの部分での最大必要量が各資源の型について分っているものと仮定する [8]．(資源使用についてそれ以上の情報が知られていることは仮定しない.) 明らかに，この“動的”MRU システムでは，(各資源に対する)任意の時点における残りの部分での最大必要量は，連鎖全体の最大必要量(これが MRU システムで知られていることが仮定されている唯一の情報である)以下である．この新しいシステムの閉鎖回避アルゴリズムを考察し，閉鎖を回避することを条件として，両システムでの資源の利用度を比較せよ．また，動的 MRU システムと資源の使い方についての完全な情報が得られていることを仮定したモデルについて資源の

利用度を比較せよ.

2.8 相互排除問題の解を得るとき, **test**$(A, a)$と表記され次のように定義されるいわゆるテスト・アンド・セット基本操作が利用できた. すなわち, **test**$(A, a)$によって, $A$ という名前の変数が 0 に設定され, もともとは $A$ が 0 でなかったのであれば制御は $a$ というラベルがついた文に移り, そうでなければ次の文が実行される. 記憶への 1 回の読取りおよび 1 回の書込みの操作を別とすると, **test**$(A, a)$が唯一の不可分操作であるという仮定の下に, 相互排除問題の解に至る効率的なアルゴリズムを作り, 適当な通信用の公共的変数を定義せよ.

2.9 $n_1$ 個および $n_2 = (n - n_1)$ 個の節点から成る互いに素な節点の集合を持つ 2 部の先行グラフ $G$ を考える. $G$ が持ち得る最大の枝の数はどれだけか. この数は, $n_1 = n_2 = n/2$ のときに最大の $n^2/4$ となることを示せ. また, $n$ 節点の先行グラフは $n^2/4$ を超える枝を持ち得ないことを示せ.

2.10 $n$–タスク・システム $C = (\mathcal{T}, \lessdot)$を考える. $\lessdot$は, $n \times n$ のブール行列 Z で表現され, Z の要素は $T_i \lessdot T_j$ のとき, しかもそのときにのみ $Z_{ij} = 1$ と定義されるものと仮定する. $C$ の先行グラフ $G$ を見出す能率の良いアルゴリズムを設計せよ. $(T_i, T_j) \in G$ であるときかつそのときにのみ $Z'_{ij} = 1$ であるような行列 Z′ を作る方法で設計せよ. (Moyles と Thompson [20] 参照.)

グラフ $G$ が分っていて, $\lessdot$を算出したいものと仮定する. Z′ が与えられているものとして, Z を見出す能率の良いアルゴリズムを設計せよ. (Warshall [21] 参照.)

2.11 $G$ は節点集合が $\mathcal{T} = \{T_1, \cdots, T_n\}$ である先行グラフであるとする. $(T_{i_j}, T_{i_k}) \in G$ ならば $j < k$ であるという性質を持つ $\mathcal{T}$ の順序付け$(T_{i_1}, T_{i_2}, \cdots, T_{i_n})$が存在することの構成的な証明を与えよ. [$G$ からこのような順序付けを作る過程は, **位相的分類**(topological sorting)と呼ばれている(たとえば Knuth [15] を見よ).]

2.12 この問題と次の問題では, 並列性についてのある "言語上の側面", すなわち, プログラミング言語で並列タスクを表現する方法を考える. より詳しい議論については, Dijkstra[4], Dennis および Van Horn[17] と Wirth[11] を参照せよ.

文

$$\textbf{fork}(L1,\ L2)$$

を実行すると，新しいタスクが $L2$ の文から開始され，もとのタスクは $L1$ の文から続行する．文

$$\textbf{join}(x,\ L1)$$

を実行すると，動作 $x \leftarrow x-1$ が行われ，$x>0$ ならばタスクは終了し，$x=0$ ならば $L1$ の文へ続く．変数 $x$ はプログラマによって初期化されるか，あるタスクで計算される．

**a.** あらゆるタスク・システム $C=(\mathcal{T},\ <)$ は，**fork-join** 構造を用いて実現できることを示せ．

**b.** **fork-join** 構造を用いたプログラムで，タスクの半順序システムとして表現できないものが存在することを示せ．特に，**fork-join** 構造を含むループおよび条件文で，いかなるタスクの半順序システムとも論理的に等価にはなり得ないものがあることを示せ．

2.13 下に基本操作文の組を三つ示す．

| 組1 | 組2 | 組3 |
|---|---|---|
| **lock** $x$ | **disable** | **wait**$(s)$ |
| **unlock** $x$ | **enable** | **send**$(s)$ |
| **fork**$(L1,\ L2)$ | **block** | |
| **join**$(y,\ L1)$ | **wakeup**$(j)$ | |

**disable** は割込みを禁止し，**enable** は改めて割込みを可能にするものと仮定する．**wait** および **send** の操作は第2章に定義されており，**fork** と **join** の操作は問題2.12で定義されている．$x$ を2値変数とすると，**lock** と **unlock** は次のように定義される．

$$\textbf{lock}:\ L:\ [\textbf{if}\ x=1\ \textbf{then}\ x \leftarrow 0\ \textbf{else goto}\ L]$$
$$\textbf{unlock}:\ [x \leftarrow 1]$$

ここで，大括弧内の動作は不可分な系列を構成しているものとする．タスク $j$ による **block** の操作は，タスク $j$ を停止させ，他のタスクが対応する **wakeup**$(j)$ を実行するまでの間のプロセッサを解放することを目論む．**wakeup**$(j)$ を実行したとき，タスク $j$ は **block** に続く次の文から再開

される．割込みが禁止されているときであれば **block** によって割込み可能になる必要があり，かつ，**block** および **wakeup**$(j)$は不可分な操作である必要がある．

**a.** 各タスク $i$ には，**起床待ちスイッチ**(wakeup-waiting switch) $s_i$ が附随していなければならないものと仮定する．操作 **wakeup**$(i)$は $s_i$ を**1**に設定する．タスク $j$ では，**block** は $s_j$ を調べ，$s_j=0$ ならば停止し，そうでなければ停止しないで $s_j$ を0に設定する．起床待ちスイッチを省くと **block** および **wakeup** についてどのように正しくない動作が起り得るかを示せ．

**b.** 相互排除を実現するための組**2**と組**3**の等価性を考察せよ．

**c.** 組**1**と組**2**が等価であるか否かを示せ．

2.14 **閉じた**タスクシステムへの一般化をいくつか考えてみよう．図 **P** 2.14(**a**)～**P** 2.14(**d**)で与えられる特別なタスクを次のように定義する．

**Y字子**(Wye)：開始して直ちに終了し，図に示した先行関係に従って後続タスクを動作可能にする．

**結合子**(Junction)：直接の先行タスクが両方とも終了したとき開始し，直ちに終了する．

**決定子**(Decider)：開始して条件 $A$ を検査し，$A$ が真ならば後続タスク $T_1$ を開始し，$A$ が偽ならば後続タスク $T_2$ を開始する．$T_1$ と $T_2$ のうち**一つだけ**が開始されることに注意

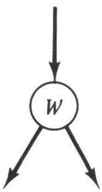



図 **P** 2.14(**a**)

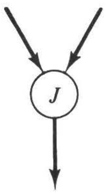



図 **P** 2.14(**b**)

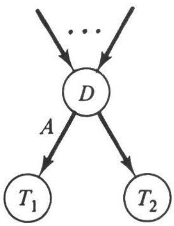



図 **P** 2.14(**c**)

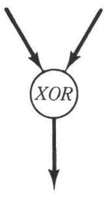



図 **P** 2.14(**d**)

せよ．

**XOR 子**(XOR)：直接の先行タスクのどちらか一方が終了したとき開始し，直ちに終了する．

**a.** $C_1$ と $C_2$ は閉じたタスク・システムであると仮定する．新しいシステム[**図 P 2.14(e)**]が決定性であるのは，$C_1$ と $C_2$ が決定性であり，かつ，$C_1$ と $C_2$ の双方で読取り専用(read-only)である記憶セル以外のセルに共通部分がないとき，かつそのときに限ることを証明せよ．

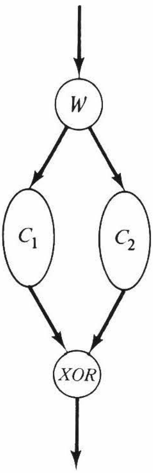



**図 P 2.14(e)**

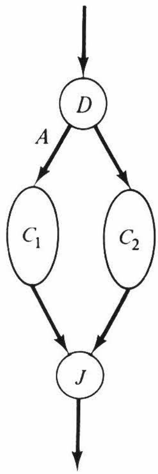



**図 P 2.14(f)**

**b.** $C_1$ と $C_2$ は閉じたタスク・システムであるとする．新しいシステム[**図 P 2.14(f)**]が決定性であるのは，$C_1$ と $C_2$ の双方が決定性であるとき，かつそのときのみであり，$C_1$ と $C_2$ の記憶セルが互いに素である必要はないことを証明せよ．条件 $A$ は，$C_1$ と $C_2$ の記憶セルの和集合内の値に関するあるブール関数である．この構造は，“**if** $A$ **then** $C_1$ **else** $C_2$” 構造に類似していることに注意せよ．

**c.** $C$ は閉じたタスク・システムであると仮定する．新しいシステム[**図 P 2.14(g)**]が決定性であるのは，$C$ が決定性であるとき，かつそのときのみであることを証明せよ．条件 $A$ は，$C$ の記憶セルの上で定義されたブール関数である．この構造は，“**while** $A$ **do** $C$” 文に類似していることに注意せよ．

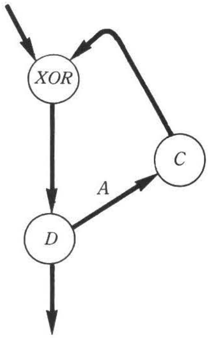



図 P 2.14(g)

**d.** 個々のタスクに構造 $A$, $B$, および $C$ を有限回適用して得られる全ての閉じたシステムから成るタスク・システムのクラス $\mathcal{C}$ を定義したとする．$\mathcal{C}$ に属するあらゆるシステムが決定性になる条件について論じよ．

2.15 図 2.5-2 には，丁度 $(n-1)$ 個の信号器を用いて，閉じた $n$-タスク・システムを実現する方法が示されていた．

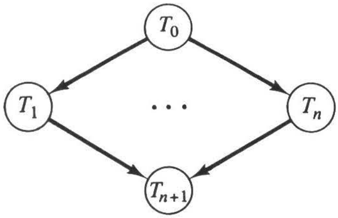



図 P 2.15

**a.** 図 P 2.15 のタスク・システムが，丁度 2 個の信号器を用いて実現される方法を示せ．

**b.** タスク・システムの実現の複雑さの尺度とは，信号器の数，**wait** 操作文の数，および **send** 操作文の数の総和であると定義する．図 2.5-2 に従って実現される閉じた $n$-タスク・システムの複雑さは，先行グラフの枝の数を $e$ としたとき，$2(n-1)+e$ であることを示せ．上記の **a.** の解の複雑さも同じであることを示せ．$n$ タスクの連鎖の複雑さは，少なくとも $3(n-1)$ であることに注意しよう．

**c.** 複雑さが $2(n-1)+e$ より小さい閉じた $n$-タスク・システムの実現は可能であろうか．

**2.16** 信号器の待ち行列の規律は後着順であると仮定する．それにもかかわらず，$n$ 個の循環タスクの集合に対するある際どい部分の**安全な**相互排除を実現したい．ここで“安全な”とは，他のタスクが勝手な回数だけ循環することにより，際どい部分に入ることを無定限に阻止されるようなタスクが一つも存在しないということを意味する．たとえば，$T_3$ が2番目に到着し，$T_1$ と $T_2$ が交互に際どい部分を通って循環するならば，$T_1$ と $T_2$ は $T_3$ を無定限に阻止できる．

葉が $n$ 枚の2進木を考え，その内部節点(および根)の各々が異なった信号器でラベル付けされているものとする．タスク $T_1, \cdots, T_n$ はある際どい部分を使うことを望んでいる．葉 $j$ から根への道は信号器 $s_{j1}, \cdots, s_{jp_j}$ でラベル付けされていると仮定する．際どい部分を使うためのタスク $T_j$ 内のプログラミングは次のとおりである．

**wait**$(s_{j1})$; $\cdots$; **wait**$(s_{jp_j})$
際どい部分の使用
**send**$(s_{jp_j})$; $\cdots$; **send**$(s_{j1})$

**a.** この解が安全であることを示せ．

**b.** 木の構造がタスク・システムの動作速度に与える影響について論じよ．

## 参考文献


1. BERNSTEIN, A. J., “Analysis of programs for parallel processing.” *IEEE Trans. Comp. EC-15*, 5 (Oct. 1966), 757–762.
2. BAER, J. L., “A survey of some theoretical aspects of multiprocessing.” *Computing Surveys 5*, 1 (March 1973).
3. COFFMAN, E. G., JR., M. J. ELPHICK, and A. SHOSHANI, “System deadlocks.” *Computing Surveys 2*, 3 (June 1971), 67–78.
4. DIJKSTRA, E. W., “Cooperating sequential processes,” in *Programming Languages* (F. Genuys, ed.). Academic Press (1968), 43–112.
5. SHOSHANI, A., and E. G. COFFMAN, JR. “Sequencing tasks in multi-process, multiple resource systems to avoid deadlocks.” *Proc. 11th Symp. Annual Switching and Automata Theory* (Oct. 1970), 225–233.
6. HOLT, R. G., “On deadlock in computer systems.” Ph.D. thesis, Dept. of Computer Science, Cornell Univ., Ithaca, N.Y., TR-71-91 (1971).
7. HEBALKAR, P. G., “A graph model for analysis of deadlock prevention in systems with parallel computations.” *Proc. IFIP Congress* (Aug. 1971), 168–172.

8. SHOSHANI, A., "Detection, prevention, and recovery from deadlocks in multiprocess multiple resource systems." Ph.D. thesis, Dept. of Electrical Engineering, Princeton Univ., Princeton, N.J. (Oct. 1969).

9. HAVENDER, J. W., "Avoiding deadlock in multitasking systems." *IBM Sys. J. 7*, 2 (1968), 74–84.

10. HABERMANN, A. N., "Prevention of system deadlock." *Comm. ACM 12*, 7 (July 1969), 373–377.

11. WIRTH, N., "A note on program structure for parallel processing." *Comm. ACM 9*, 5 (May 1966), 320–321.

12. BREDT, T. H., and E. J. MCCLUSKEY, "Analysis and synthesis of control mechanisms for parallel processes," in *Parallel Processor Systems, Technologies and Applications* (L. C. Hobbs et al., eds.). (1970), 287–296.

13. KNUTH, D. E., "Additional comments on a problem in concurrent programming." *Comm. ACM 9*, 5 (May 1966), 321–323.

14. FLOYD, R. W., "Assigning meanings to programs." *Proc. Symp. Appl. Math.* (1967), 19–32.

15. KNUTH, D. E., *The Art of Computer Programming* (Vol. I). Addison-Wesley, Reading, Mass. (1968).

16. DIJKSTRA, E. W., "Solution of a problem in concurrent program control." *Comm. ACM 8*, 9 (Sept. 1965), 569.

17. DENNIS, J. B., and E. C. VAN HORN, "Programming semantics for multiprogrammed computations." *Comm. ACM 9*, 3 (Mar. 1966), 143–155.

18. BRINCH HANSEN, P., "The nucleus of a multiprogramming system." *Comm. ACM 13*, 4 (Apr. 1970), 238–241, 250.

19. SPIER, M. J., and E. I. ORGANICK, "The Multics interprocess communication facility." *Proc. 2nd ACM Symp. Op. Sys. Princ.* (Oct. 1969), 83–91.

20. MOYLES, D. M., and G. L. THOMPSON, "An algorithm for finding a minimum equivalent graph of a digraph." *J. ACM 16*, 3 (July 1969), 455–460.

21. WARSHALL, S., "A theorem on Boolean matrices." *J. ACM 9*, 1 (Jan. 1962), 11–12.

22. KARP, RICHARD M., and RAYMOND E. MILLER, "Properties of a model for parallel computations: determinacy, termination, and queueing." *Siam Journal of Applied Math 14*, 6 (Nov. 1966), 1390–1411.

23. COMMONER, F., A. W. HOLT, S. EVEN, and A. PNUELI, "Marked directed graphs." *Journal of Computer and System Sciences 5*, 5 (Oct 1971).

24. KARP, R. M., and R. E. MILLER, "Parallel program schemata." *Journal of Computer and System Sciences 3*, 2 (May 1969), 147–195.

25. Bruno, J. L., E. G. Coffman, Jr., and W. H. Hosken, "Consistency of synchronization nets using P and V operations." *Proc. 13th Symp. on Switching and Automata Theory* (Oct. 1972), 71–76.

26. Keller, R. M., "On maximally parallel program schemata." *Proc. 11th Symp. on Switching and Automata Theory* (Oct. 1970), 33–50.

27. Eisenberg, M. A., and M. R. McGuire, "Further comments on Dijkstra's concurrent programming control problem." *Comm. ACM 15*, 11 (Nov. 1972), 999.

28. de Bruijn, "Additional comments on a problem in concurrent programming control." *Comm. ACM 10*, 3 (Mar. 1967), 137–138.

# プロセッサ・スケジューリングの決定性モデル

# 3

## 3.1 はじめに

第1章で見たように，多重プロセッサ・システムには数々の利点があり，そのうち，大多数の利用者にとって最も重要なものの一つとして並列プログラミング，すなわち，一連の計算の独立した部分を並行して実行することによって計算時間が短縮される可能性がある．このことは実時間的な応用や，実験データの整理を含む数々の応用で見られる長時間の計算において，その結果を単一プロセッサ・システムが可能にするよりももっと早く得る必要がある場合にとりわけ重要である．これらの応用に対して最大の性能が得られるように多重プロセッサ構成を開発する最良の方法が本章の主な関心事である．

本章のはっきりした目標は，スケジュールされるべきタスク・システム群[†](あるいはこれらのシステムの見積り)が前もって知られており，かつ，スケジュールするときには全てのタスク・システムが同時に取扱えるようになっているという仮定の下に，多重プロセッサ・システムのスケジューリングのための効率の良いアルゴリズムを示すことである．第2章の基本モデルを拡張して，各タスクの実行時間(正数)が指定されているという仮定を含むようにしよう．タスクの半順序集合という仮定に制約が内在しているけれども，この領域における理論的な結果の大多数は，これらの構造あるいはその特別な場合に適合する．

本章の以下の五つの節では，多重プロセッサ・システムが単に $m(\geq 2)$ 個の同一かつ独立なプロセッサから構成されていると定義されている場合のスケジューリング問題を考察する．3.7節では，異なった機能を持つ異種のプロセッサを含むシステムについて，二，三の既知の結果を示す．これらの節に示され

† **訳注**　特に3.7節では，タスク・システムの集合が対象にされている．

ているアルゴリズムの基本的な目標は，タスク・システムが最短時間で実行されるようにそのタスク・システムをスケジュールすることである．3.8節では別の重要な目標を考察するが，主に単一プロセッサの仮定下の場合についてである(この仮定は，それに関連した問題の理論的な取扱いがうまくいくために必要である)．本節の残りの部分では，$m(\geq 2)$個の同一プロセッサ上でのスケジューリング問題に関連した概念と基本的な着想を展開する．

$m$台のプロセッサおよび与えられた先行グラフに対する**スケジュール**(schedule)または**割当て**(assignment)は，時々刻々に各プロセッサでなされるべき仕事を記述したものである．スケジュールは，与えられた先行関係を乱したり，どの時間間隔をとっても一つのタスクに二つ以上のプロセッサを割当てたりしてはならないし，タスクの実行時間に等しいだけのプロセッサ時間を配分しなければならない†．スケジュールを明確に示す最も単純な方法は，ガント図(Gantt chart) [1]を用いることである．これは，各プロセッサごとの時間軸から成っており，その時間軸は間隔に区切られ，処理中のタスクの名前がラベルとして書かれている．**図3.1-1(a)**にグラフ$G$(より正確には，グラフ$G$を持つタスク・システム)を示し，**図3.1-1(b)**には，二つのプロセッサ$P_1$と$P_2$による$G$の一つのスケジュールを示す．スケジュールを表すのに記号$\mathcal{S}$を用

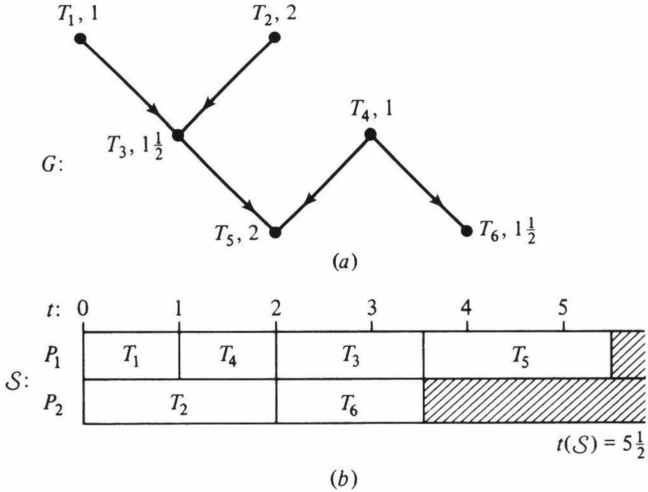



**図3.1-1**　グラフとガント図の例

† **訳注**　この条件を満たすスケジュールは正当(valid)であるという．

い，$\mathcal{S}$ の長さ，すなわち，$\mathcal{S}$ における最後の終了時刻を表すのに記号 $t(\mathcal{S})$を用いる．プロセッサの遊休時間を表すには図中で斜線を用いる．また，間隔 $[0, t(\mathcal{S})]$に含まれる遊休期間の間，プロセッサは空タスク(empty task)$\phi$ を実行しているということがある．記号 $\tau_i$ はタスク $T_i$ の実行時間を表す．

各タスクに附随する実行時間には，少なくとも三つの現実的な解釈がある．まず第一に，そのタスクの既知かつ決定性(deterministic)の実行時間を表しているかも知れない．この場合，作成されたスケジュールは明確な解釈を持つ．第二に，タスクの実行時間の可能な最大値を示すこともあろう．これらの値を使ったときのスケジュールは，結局のところ最悪事例解析(worst-case analysis)であり，遵守されなければならない厳しい納期がある場合の"難しい"実時間の問題に適用できる．この解釈の下では，一度スケジュールが作成されてしまうと，プログラムの実行はプロセッサの割当てに関する次の二つの基本的な方策のいずれかに従って制御される．

1. プロセッサ $P_j$ がある時間間隔をタスク $T_i$ 上で動作するように割当てられたとすると，制御機構はその全時間間隔にわたって $P_j$ を $T_i$ のためにとっておく．与えられた走行について $T_i$ の実行がその最大値まで至らないで終了した場合は，プロセッサは遊休状態のままにされるか，ある種の背景の仕事(background work)に使われる．
2. 制御機構はタスク・システム全体の実行時間が短くなるように，個々の走行の実行時間が(最大値に比べて)短くなったことを利用するように作られる．そのような制御機構の一つが3.5節に述べられている．

第三に，与えられたタスクに附随している重みは，確率変数として見たそのタスクの実行時間の期待値でもよい．各タスクにその実行時間の期待値に等しいプロセッサ時間を割当てるスケジュールは，次のような方法で使うことができる．このスケジュールは，各タスクそれぞれにプロセッサを割当て，同じプロセッサを割当てられたタスク同志の間の順序を定める．この割当てと順序が固定されたと仮定し，同じプロセッサ上で先行している全てのタスクが完了し，かつ，グラフにおけるその全ての先行者が完了すると，ただちにタスクは開始されると仮定する．次に示すような先行関係 $T_i < T_j$ の全てをグラフ $G$ に付け加えた新しいグラフ $G'$ を定義する．すなわち，$G$ において $T_i$ と $T_j$ が互いに独立であり，ある同一のプロセッサに割当てられているタスクの順序では $T_i$ が $T_j$ に先行して出現するとき，$T_i < T_j$ とする．このスケジュールの平均の長さを楽観的に評価すると，$G'$ における最長の道の長さである．そ

の合理的な発見的手法は，与えられた実行時間が期待値であると仮定して，対応する割当てと順序を使って最短のスケジュールを見出すことから構成されよう．

以下，引続いて定義する全てのスケジューリング問題に対し，最適スケジュールの探索には有限個のスケジュールだけを考慮すればよい．つまり，これらの問題は全て，数え上げ(全ての可能性の吟味)によって効果的に可解(effectively solvable)である．多くのスケジューリングの問題には，動的計画法，分岐限定法(branch and bound)や数理計画法によるその他の手法[2]が適用されてきた．これらは，本質的には数え上げの手法であって，発見的手法の研究における比較の標準を作るために効果的に用いられてきた．ところで，我々の目的にとっては，タスク・システムをスケジュールするための"効率の良い"アルゴリズムは，本質的に数え上げ法でないもの，たとえばタスクの数に対して指数的になるのではなく多項式的に変化する手数で済むものを意味する．

以下では，**横取り権のある**(preemptive)† 場合と**横取り権のない**(non-preemptive)† 場合の両方のスケジューリング規律に関心を払う．横取り権なしスケジューリングでは，一旦始まったタスクは完了するまで実行されなければならない．横取り権ありの規律下では，タスクは完了する前に割込まれ，他のタスクのためにプロセッサを取上げられるかも知れない．もちろん，横取りされたタスクは，割込まれた点から再開され，究極的には完了するに至る十分なプロセッサ時間が与えられるということが依然として仮定されている．

実際的な見地からすれば，制限付きの横取り権ありの規律(たとえば，引続く横取り間の時間間隔に下限を設けるとか，単位時間内に行い得る横取りの回数に上限を置くなど)を考えるのは有効であろう．しかし，ここでは制限なしの規律のみを取扱う．

横取り権ありの規律では，実行できる状態のタスクがあるにもかかわらずプロセッサに遊休期間を導入することは，決して利益にならないということに特に注意しよう．横取り権なしの規律ではそれが当てはまらないことは**図3.1-2**のグラフとガント図で示される．(最適)スケジュール $S$ では，$\epsilon$ から $2\epsilon$ までの間の $P_3$ が遊休である時に，$T_7$ は引続いて割当て可能になっていることに注意しよう．この $T_7$ を $t=\epsilon$ で $P_3$ に割当てると，より長いスケジュール $S'$ になってしまう．

---

† **訳注**　"preemptive" および "non-preemptive" は，本章ではこのように訳すが，第4章以降では少し異なった訳語が用いられる．

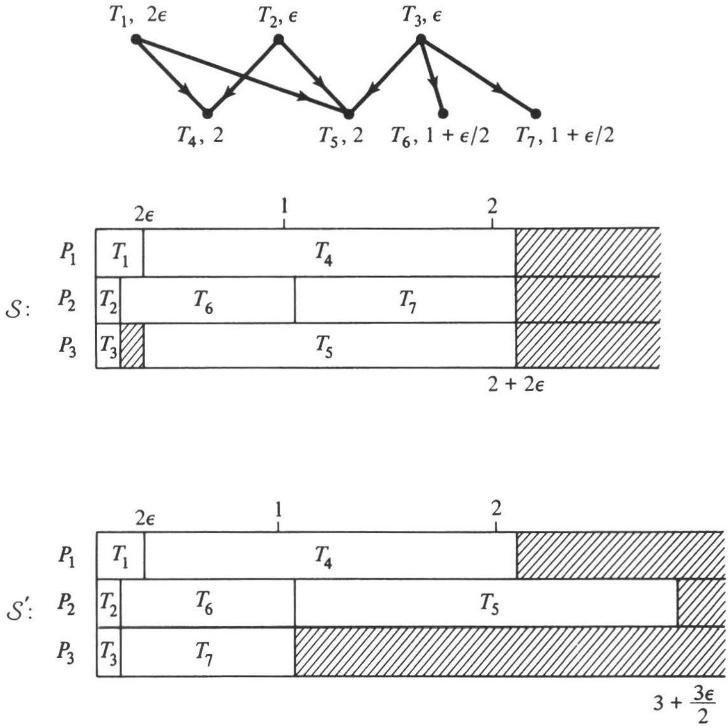



図 3.1-2　遊休時間の導入

## 3.2 2-プロセッサ・システムに対する最適スケジュール[3]

先行関係の構造が任意であり，最適の横取り権なしスケジュールが能率良く計算できる現在知られている唯一の場合は，プロセッサ 2 台で，かつ，全てのタスクの実行時間が等しいという仮定を必要とする†．このシステムの最も直接的で実際的な応用は，スケジューリングに先立ってほぼ同じ大きさのタスクおよび対応する半順序に計算を分割できるような状況に対してである．次節の木-スケジューリングの結果と同様に，3.6 節では，2-プロセッサの結果の重要性は主として最適の横取り権ありスケジュールとの関連性にあることが分る．

2-プロセッサ問題の解には，グラフ $G$ の各タスクに正整数を対応づけるラベル付けアルゴリズムが含まれている．スケジューリングのアルゴリズム

† 一般の問題の計算の複雑さに関する扱いについては，文献[19]参照.

は，それによって定まるタスクのラベルを用いて定義される．ラベル付けアルゴリズムの定義を簡単にするため，次の定義を導入する．正整数の減少する系列同志の間に次のような線形順序を付ける．$N=(n_1, n_2, \cdots, n_t)$および$N'=(n'_1, n'_2, \cdots, n'_{t'})$をそのような2系列とする[††]．すなわち，$1 \leq i < t$および$1 \leq j < t'$に対し，$n_i > n_{i+1}$および$n'_j > n'_{j+1}$である．次のいずれかが成り立つとき$N < N'$であるという．

1. $t=0$(すなわち，$N=(\ )$)，かつ，$t' \geq 1$.
2. ある$i (1 \leq i \leq t)$が存在し，$1 \leq j \leq i-1$を満たす全ての$j$に対して$n_j = n'_j$かつ$n_i < n'_i$.
3. $t < t'$かつ$n_j = n'_j (1 \leq j \leq t)$.

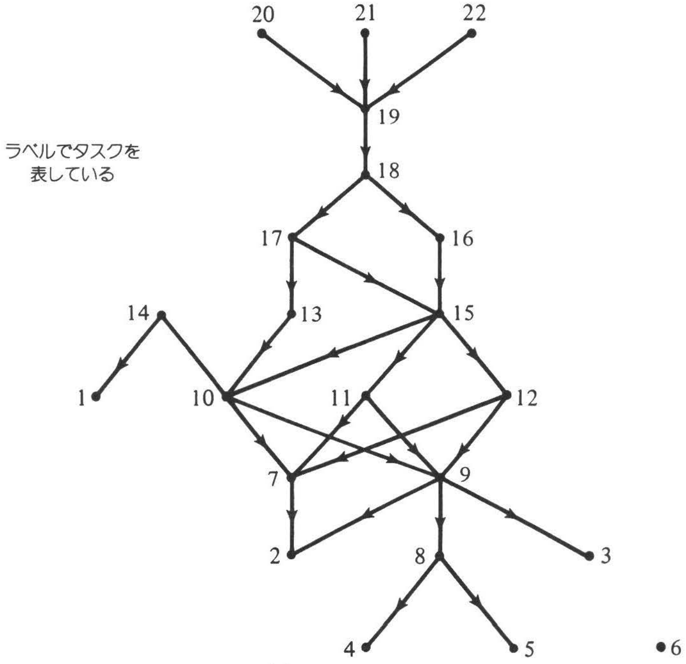
たとえば，$(7, 5, 3, 2) < (7, 5, 4, 1)$，$(4, 3, 1) < (5, 1)$，$(7, 2) < (7, 2, 1)$，$(\ ) < (3)$である．

$G$のタスク数を$n$で表す．ラベル付けのアルゴリズムは各タスク$T$に整数のラベル$\alpha(T) \in \{1, 2, \cdots, n\}$を割付ける．写像$\alpha$は次のように帰納的に定義される．$T$の直接の後続者の集合を$S(T)$で表す．

1. $S(T_0)=\phi$であるような任意のタスク$T_0$を選び，$\alpha(T_0)=1$と定義する．
2. ある$k \leq n$に対して，整数$1, 2, \cdots, k-1$が既に割付けられていると仮定する．$S(T)$が空であるか，または，$S(T)$の全要素に対する$\alpha$が既に定義されたタスク$T$の各々について，集合$\{\alpha(T') | T' \in S(T)\}$の要素を順に並べて得られる整数の減少系列を$N(T)$で表す．これらのタスクのうち少なくとも一つの$T^*$は，これら全てのタスク$T$に対して$N(T^*) \leq N(T)$を満たしているはずである．そのような$T^*$を一つ選び，$\alpha(T^*)=k$と定義する．
3. $G$の全タスクがある整数を割付けられるまで，2.の割付けを繰返す．

**図3.2-1**は，正しくラベル付けを行ったグラフの例を示す．たとえば，ラベル10, 11, 12を持ったタスクには，これらの整数を任意の順に割付けても良いことに注意しよう．上記のラベル付けアルゴリズムに基づくスケジューリング・アルゴリズムを次のように定義する．**プロセッサが空いた時はいつでも，先行タスクの全てが既に実行済みであり，かつプロセッサがまだ割当てられていないタスクの中で最大のラベルを持つものに割当てる**．簡単のため，一般性

†† **訳注**　$t=0$のとき，$N=(\ )$(空系列)とする．




(*a*)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $P_1$ | 22 | 20 | 19 | 18 | 17 | 15 | 12 | 10 | 9 | 8 | 5 | 2 | |
| $P_2$ | 21 | 14 | 6 | 1 | 16 | 13 | 11 | | 7 | 3 | 4 | | |

*A*-スケジュール

(*b*)

**図 3.2-1** *A*-スケジュールの例

を失うことなく, 2-プロセッサ・システムにおいて $P_1$ と $P_2$ が同時に利用可能になったときは, 常に $P_1$ の方が先に上のアルゴリズムによってタスクを割当てられることにする. (プロセッサの数が任意の場合にも同様にこの取決めを採用しよう). 上のアルゴリズム(アルゴリズム A と呼ぶ)で作ったスケジュールを **A-スケジュール**と呼ぶことにしよう. 一般性を失うことなく, タスクの実行時間を時間の単位として採用する. プロセッサ数 $m=2$ でタスクの実行

時間がみな等しい場合は A–スケジュールは最適であるということのかなり難しい証明を吟味する前に，ラベル付けのアルゴリズムとそれに対応する A–スケジュールの重要な性質をいくつか議論しよう．

いま考察しているスケジューリング問題をある程度考えてみると，ラベル付けアルゴリズムは次の性質を持っているということが直観で分る．

1. $T$ が $T'$ よりもレベルが高ければ $\alpha(T) > \alpha(T')$ である．すなわち，$T$ は $T'$ よりスケジューリング優先度が高い．実例を図 3.2–1 で調べよ．
2. $S(T') \subset S(T)$ ならば，やはり $\alpha(T) > \alpha(T')$ である．たとえば，図 3.2–1 で，ラベルが 15 のタスクは，ラベルが 13 のタスクより高い優先度(ラベル)を持っていることが分る．

これまで述べてきたラベル付けアルゴリズムが性質 2 を持つことは容易に分る．さらに，性質 1 もこのラベル付けアルゴリズムの性質であることを示すのはそれ程難しくない．$G$ のレベルの数についての帰納法を使って直接，証明することができる(問題 3.1 を見よ)．

一般に，タスク $T$ の実行が済むと数多くのタスクの実行が直ちに可能になるとき，$T$ には高いスケジューリング優先度を，場合によってはより高いレベルにある他のタスクよりも高い優先度さえも，与えるべきであると考えるのは自然であろう．しかし，スケジュール $A$ では，性質 2 の条件下でなければそれは実現されず，その条件下だけで十分であるのは $m = 2$ であるという前提に際どく依存している．たとえば図 3.2–2 のグラフを調べてみよう．$m = 2$ の場合は $A$–スケジュールが最適であることは容易に確かめられるが，$m = 3$ の場合は図に示したように最適にはならない．明白な理由は，直接の後続者が多数あることから当然与えられるべき十分高い優先度を $T_5$ が与えられていないことである．

最適のラベル付けアルゴリズムを定義するのに性質 1 と 2 で十分であると考えたくなるが，それは間違っている．問題 3.4 は，読者が先へ進む前に考えて欲しい反例を与えてくれる．

実行が等(単位)時間であるから，$P_1$ も $P_2$ も同時にタスクの実行に使えるようになる．時刻 $t$ で $P_1$ と $P_2$ が新しいタスクの実行に使えるようになったとし，$T$ が時刻 $t$ ではまだ実行されていないタスクに対応するラベルの中で最高のラベルを持った未実行のタスクであると仮定する．$T' < T$ であるような全ての $T'$ に対し，アルゴリズム A で作られたラベル付けは $\alpha(T') > \alpha(T)$ である．したがって，時刻 $t$ では $T$ の全ての先行者は実行済みのはずであり，

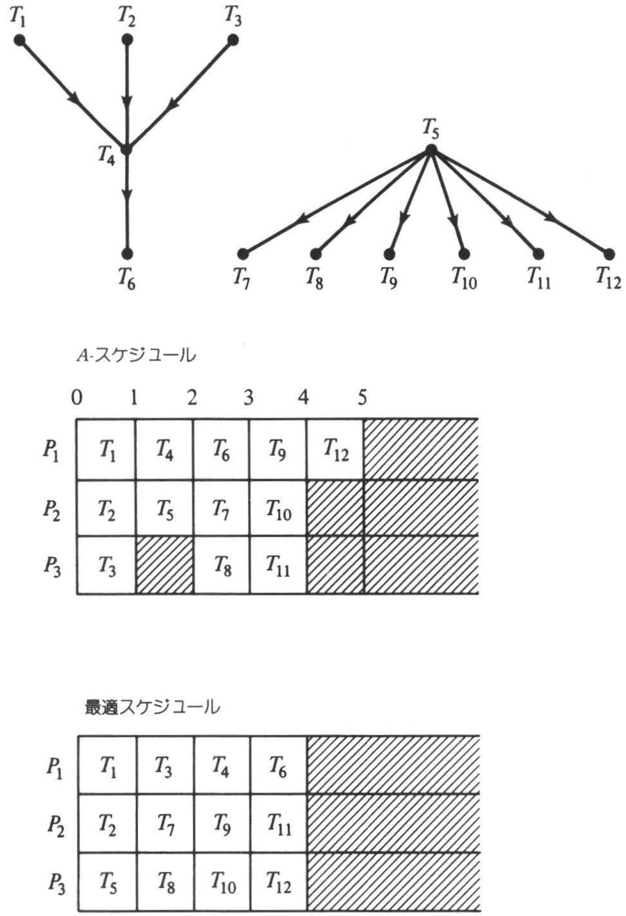



**図 3.2-2** $m=3$ に対する反例

$T$ は実行できる状態になっている．$A$-スケジュールの定義から，$P_1$ と $P_2$ は常に最高のラベルを持った未実行のタスクを実行しようとするし，取決めにより $P_1$ が $P_2$ より前にタスクを割当てられることから，$T$ は時刻 $t$ から始まる単位時間間隔の間 $P_1$ によって実行されるはずである．このことから次の性質が成立つのであるが，$A$-スケジュールが最適であることを証明する時に参照し易いように補題の形にしておこう．

**補題 3.1**

$G$ に対する $A$–スケジュールで，タスク $T$ が実行を開始する時刻を表す非負整数を $t(T)$ と定義する．$T$ が $P_1$ で実行され，かつ $t(T) \leq t(T')$ ならば $\alpha(T) > \alpha(T')$.

なお，前記により，$A$–スケジュールの最後のタスクの終了時刻より前に $P_1$ が空くことは決してないことにも注意しよう．ここで主な結果を次に示す．

**定理 3.1**

タスクの実行時間が全て同じである任意のグラフに対し，$m = 2$ ならば，$A$–スケジュールの長さは最短である．

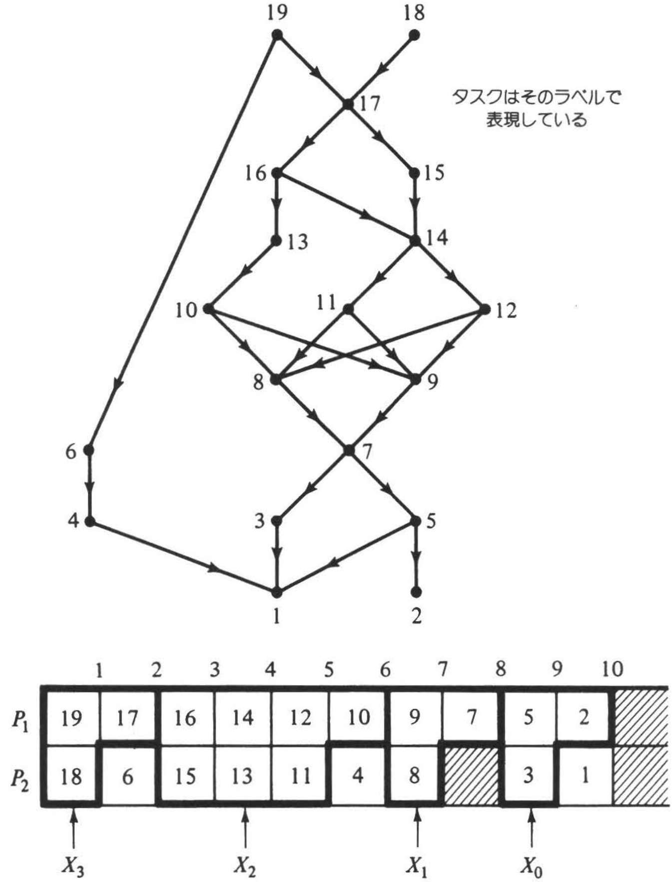
**証明**　いくつかの定義をすることから始めよう．$G$ のタスクは $A$–スケジュール $\mathcal{S}$ を使って実行されるものと仮定する．$P_2$ が時刻 $t$ から時刻 $(t + 1)$ まで空いているとき，$P_2$ は空タスク $\phi$ を実行しているといい，$\alpha(\phi) = 0$ と定義する．タスク $V_i$ および $W_i$ を次のように帰納的に定義する．

1. $V_0$ は，$t(V_0) = t(\mathcal{S}) - 1$ を満たし，$P_1$ で実行されるタスクであると定義する(すなわち，$V_0$ は $P_1$ で実行される最後のタスクである)．同様に，$W_0$ は $t(W_0) = t(\mathcal{S}) - 1$ を満たし $P_2$ で実行される(場合によっては空の)タスクであると定義する．
2. 一般に，$k \geq 1$ に対し，$W_k$ は $\alpha(T) < \alpha(V_{k-1})$，$t(T) < t(V_{k-1})$ かつ $t(T)$ は**最大**であるような(場合によっては空の)タスク $T$ であると定義する．補題3.1 より，$W_k$ は $P_2$ で実行されなければならない．$V_k$ は $t(V_k) = t(W_k)$ を満たし $P_1$ で実行されるタスクであると定義する．

$W_1$ が存在しないならば，時刻 $t(\mathcal{S}) - 1$ 以前にプロセッサが遊休であったことはなく，明らかに $\mathcal{S}$ は最適である．したがって，$W_1$ が(したがってまた $V_1$ も)存在すると仮定して良い．$1 \leq k \leq r$ に対して $W_k$ を定義できたとする．$t(V_{i+1}) < t(T) \leq t(V_i)$ かつ $T \neq W_i$ を満たすタスク $T$ の集合を $X_i (0 \leq i \leq r)$ と定義する．$V_{r+1}$ は存在しないから，$X_r$ は $t(T) \leq t(V_r)$ かつ $T \neq W_r$ を満たす $T$ の集合である．

各 $X_k$ の要素数は奇数であり，ある正整数 $n_k$ に対し，$|X_k| = 2n_k - 1 (0 \leq k \leq r)$ と書けることに注意しよう．上の定義を説明する例を**図 3.2–3** に示す．集合 $X_k$ は太線で囲んである．

定理の証明の核心は，$0 \leq k \leq r$ に対し，$T \in X_k$，$T' \in X_{k+1}$ ならば $T' <$




$\alpha(V_0) = 2$　　$\alpha(V_1) = 7$　　$\alpha(V_2) = 10$　　$\alpha(V_3) = 17$

$\alpha(W_0) = 1$　　$\alpha(W_1) = 0$　　$\alpha(W_2) = 4$　　$\alpha(W_3) = 6$

**図 3.2-3**　$V_i$, $W_i$, および $X_i$ の定義

$T$ であるという事実にある．これを $t(T)$ および $t(T')$に関する(二重)帰納法で証明を進めよう．$X_k$ の定義および補題 3.1 より，$T \in X_k$ ならば $\alpha(T) \geq \alpha(V_k)$ かつ $t(T) \leq t(V_k)$である．

まず，$T \in X_k$ を $t(T)$が最小のものである，すなわち $t(T) = t(V_{k+1}) + 1 = t(V_k) - n_k + 1$ であるとしよう．

$$\alpha(V_{k+1}) > \alpha(T) \geq \alpha(V_k) > \alpha(W_{k+1})$$

であるから $P_2$ が $T$ を時刻 $t(V_{k+1})$に実行しようとしたと考えられるが，そうなっていない．したがってその時刻には， $T$ のある先行者 $T'$ があってその実行がまだ完了していなかったに違いない．つまり，$t(T') \geq t(V_{k+1})$．しかし，$T'$が $T$ の先行者であることは $A$-スケジュールの定義から，$\alpha(T') > \alpha(T)$を意味している． $T$ は時刻 $t(T)$に実行されたのであるから， $T'$ は $T$ より前に実行されていなければならず，

$$t(T') \leq t(T) - 1 = t(V_{k+1})$$

以上により，$t(T') = t(V_{k+1})$かつ $\alpha(T') > \alpha(W_{k+1})$．$T'$ に対して可能性はただ一つしかなく，$T' = V_{k+1}$ である．つまり，$V_{k+1} \lessdot T$.

次に， 任意に固定した $j(1 \leq j \leq n_k)$に対し， $T \in X_k$ かつ $t(T) \leq t(V_k) - n_k + j$ ならば $V_{k+1} \lessdot T$ であることが示されたと仮定する． $T' \in X_k$ は $t(T') = t(V_k) - n_k + j + 1$ を満たしているとする．前と同じ議論により，

$$\alpha(V_{k+1}) > \alpha(T') \geq \alpha(V_k) > \alpha(W_{k+1})$$

が成り立つから，$T'$ は時刻 $t(V_{k+1})$に $P_2$ で実行されようとしたと考えられるが，実行可能にはなっていなかった．つまり，$T'$ のある先行者 $T''$ があって，この時刻までには実行を完了していなかったのであり，次の式(A)が成り立っていなければならない．

(A) $$t(T'') \geq t(V_{k+1}) = t(V_k) - n_k$$

また，$T'' \lessdot T'$ であるから，$t(T'') \leq t(T') - 1 = t(V_k) - n_k + j$ である．式(A)において，二つの場合が考えられる．まず，$t(T'') = t(V_k) - n_k$ であると仮定するならば，$\alpha(T'') > \alpha(T') \geq \alpha(V_k) > \alpha(W_{k+1})$であるから， $T'' = V_{k+1}$ でなければならない． したがって， 所望の $V_{k+1} \lessdot T'$ を得る． 次に，$t(T'') \geq t(V_k) - n_k + 1$ であると仮定するならば，

$$t(V_k) - n_k + 1 \leq t(T'') \leq t(V_k) - n_k + j$$

であり，また，$T'' \in X_k$ であるから， 帰納法の仮定より $V_{k+1} \lessdot T''$ であることが分る．ゆえに， 半順序 $\lessdot$ の推移性より $V_{k+1} \lessdot T'$ が得られ，二重帰納法の第1段が終了する．以上が全ての $T \in X_k$ に対して $V_{k+1} \lessdot T$ であることの証明である．

$X_k$ 内に先行者を持たない $X_k$ 内のタスクの集合を $I_k$ で表す．全ての $T \in X_k$ に対して $V_{k+1} \lessdot T$ であるから，$S(V_{k+1}) \cap X_k = I_k$ であることを知るのは難しくない†.

任意に選んだ $j(0 \leq j \leq n_{k+1} - 2)$ に対し，$t(V_{k+1}) - j \leq t(T) \leq t(V_{k+1})$ を満たす任意の $T$ が $X_{k+1}$ に属するならば全ての $U \in X_k$ に対して $T \lessdot U$ であるということが証明されたと仮定しよう．$T' \in X_{k+1}$ は $t(T') = t(V_{k+1}) - j - 1$ を満たしているとする．$T' \in X_{k+1}$ であるから $\alpha(T') > \alpha(V_{k+1})$. $N(T')$は $T'$ の直接の後続者の $\alpha$ の値の減少系列で作られていることを思い出すと，$N(T') \geq N(V_{k+1})$でなければならない．もし，$T'' \in S(T') \cap X_{k+1}$ が存在するならば，$t(T'') > t(T') = t(V_{k+1}) - j - 1$, すなわち $t(T'') \geq t(V_{k+1}) - j$ であるから，帰納法の仮定により全ての $T \in X_k$ に対して $T'' \lessdot T$ であり，推移律から全ての $T \in X_k$ に対して $T' \lessdot T$ となる．そこで，$S(T') \cap X_{k+1} = \phi$ であると仮定しよう．補題3.1より $t(T) > t(V_k)$ ならば $\alpha(T) < \alpha(V_k)$．また，$\alpha(W_k) < \alpha(V_k)$ でもある．ここで少々考えてみると，A-スケジュール，ラベル付けアルゴリズムおよびグラフ $G$ の定義から，$N(T') \geq N(V_{k+1})$かつ $S(T') \cap X_{k+1} = \phi$ ならば，$S(T') \cap X_k = I_k$ でなければならないことが分る††. このことは，全ての $T \in X_k$ に対し，$T' \lessdot T$ であることを意味している．これで二重帰納法の第2段が完了し，$T \in X_k$ かつ $T' \in X_{k+1}$ $(0 \leq k \leq r)$ならば，$T' \lessdot T$ であることの証明が完了した．

---

†訳注　先行グラフには冗長な枝がないことに注意せよ．

††訳注　[命題] $N(T') \geq N(V_{k+1})$ かつ $S(T') \cap X_{k+1} = \phi (T' \in X_{k+1},\quad t(T') = t(V_{k+1}) - j - 1,$ $0 \leq j \leq n_{k+1} - 2, k \geq 0)$ ならば，$S(T') \cap X_k = I_k$ である．

**証明**　$S(T') \cap X_k \neq I_k$ を仮定して矛盾を導こう．背理法のこの仮定を二つの場合に分けて考える．

(1)　ある $T \in I_k$ が存在して，$T \notin S(T') \cap X_k$ である場合．

$T'$ の定義から $T' \neq V_{k+1}$ であり，$I_k$ の定義から $T \in X_k$ ゆえ，$T \notin S(T')$かつ $T \in S(V_{k+1})$である．さらに二つの場合に分けて考える．

(i)　$T' \lessdot T$ が成り立つとすると，$T \notin S(T')$であることから，ある $T'' \in S(T')$が存在して $T' \lessdot T'' \lessdot T$. 命題の仮定から $T'' \notin X_{k+1}$. したがって $T'' \in X_k \cup \{W_{k+1}\}$でなければならない．しかし，ラベルの性質から明らかに $T'' \neq W_{k+1}$ であるから，$T'' \in X_k$ でなければならない．したがって，$V_{k+1} \lessdot T'' \lessdot T$. グラフ $G$ には冗長な枝がないことから，これは $T \in S(V_{k+1})$と矛盾する．

(ii)　$T' \not\lessdot T$ が成り立つとする．$N(V_{k+1}) = (n_1, n_2, \ldots, n_l)$, $N(T') = (n'_1, n'_2, \ldots, n'_{l'})$と書く．任意の$T'' \in X_k$ に対し，$V_{k+1} \lessdot T''$ であるから，減少列 $(n_1, n_2, \ldots, n_l)$ の前半は $X_k$ に属する($\lessdot$ に関して)極大なタスクのラベル($\alpha(T)$を含む)の減少列に一致する．一方，$S(T') \cap X_{k+1} = \phi$ であることから，減少列$(n'_1, n'_2, \ldots, n'_{l'})$の前半は $S(T') \cap X_k$ に属する極大なタスクのラベルの減少列に一致し，$\alpha(T)$を含まない．これは，命題の仮定 $N(T') \geq N(V_{k+1})$と矛盾する．

(2)　ある $T \in S(T') \cap X_k$ が存在して，$T \notin I_k$ である場合．

$T \in X_k$かつ $T \notin I_k$ であるから，$T$ は $X_k$ 内にある先行タスク $T''$ を持ち，$T' \not\lessdot T''$. 上記(ii)と同じ議論((ii)において $T$ が果した役割を $T''$ に担わせるとよい)により，命題の仮定と矛盾する．

■

つぎは，定理の証明に向けての短い手順である．任意のスケジュールに対し，$X_{k+1}$ 内の**全ての**タスクは，$X_k$ 内の任意のタスクが開始する前に実行されなければならない．$X_{k+1}$ は $(2n_{k+1}-1)$ 個のタスクから成っているので，それには少なくとも $n_{k+1}$ 時間単位が必要である．つまり，$G$ を実行するには，使用するスケジュールの如何を問わず，少なくとも $\sum_{k=0}^{r} n_k$ 時間単位が必要である．$t(\mathcal{S}) = \sum_{k=0}^{r} n_k$ であるから，$\mathcal{S}$ は最短であることが示されたことになり，定理の証明は完成した． ■

$m>2$ の場合は $A$-スケジュールが必ずしも最適にはならないことは既に示した(図3.2-2参照)．実行時間が全部同じとは限らない場合も，$A$-スケジュールは必ずしも最適にはならないこともまた真実である．タスクの実行時間が1単位か2単位のいずれかをとり得ると想定しよう．図3.2-4は，そのようなタスクから成り，2台のプロセッサに対する $A$-スケジュールが最適にはならな

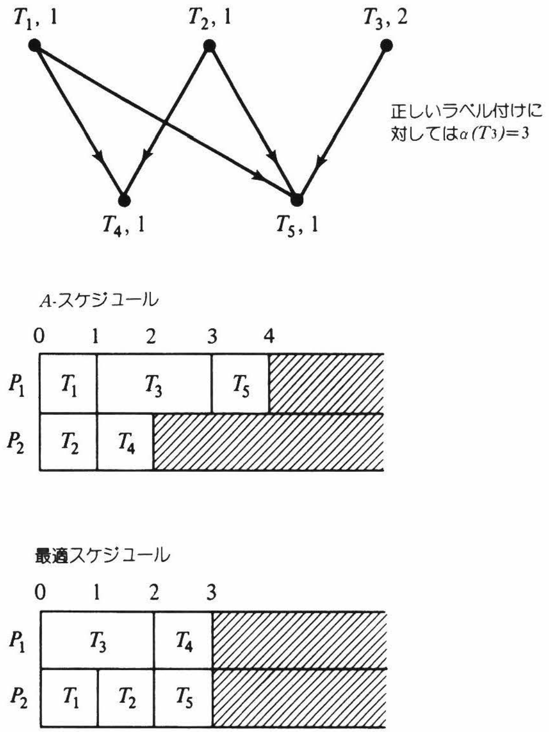



**図3.2-4**　実行時間が等しくない場合の反例

いグラフを示している．この例は，タスクのスケジューリング優先度はその実行時間，あるいはより一般的には，その与えられたタスクを含み実行時間の総計が最大になるような道の総実行時間に依存して定められるべきであることを示している．

## 3.3　木構造の先行グラフに対する最適スケジュール

本節では，先行グラフが木構造になっている特別かつ重要な場合を考察しよう．タスクのスケジュールに利用できるプロセッサの数は任意であると仮定する．全てのタスクの実行時間が単位時間であるという仮定および3.2節のその他の仮定は，本節でも引継がれる．

本節で取上げる先行関係の構造は(有向の)単一根の木であり，根を除く各節点は正確に一本の出射枝を持つ．根にはそのような枝はない．これから考察する木は，任意に与えられた節点(タスク)の直接の先行者同志には順序が仮定されていないという意味で順序なしである．このような木の一例を図3.3-1に示す．

木に対する最適スケジュールを求める簡単なアルゴリズムが考案されている[4]．このアルゴリズムは，直観的には大いに心に訴え易くこのスケジューリング問題を考察する誰もが全く同様なことを思い浮かべそうなものではあるけれども，それが最適であることの証明は予期したよりもずっと難しい．このア

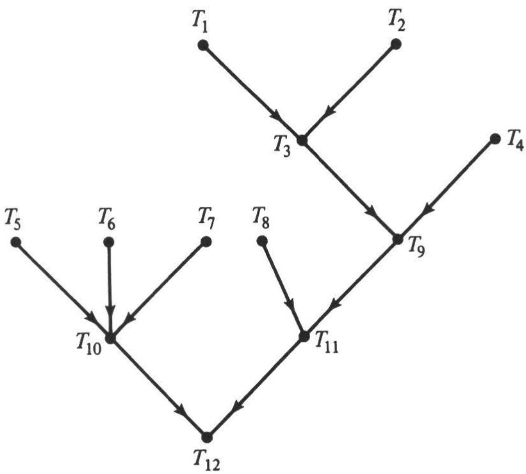



**図3.3-1**　先 行 木

ルゴリズムの記述とその最適性の証明を次に示す．

以下に記述されるアルゴリズムでは，最初に，与えられた木の中にあるタスクの集合が既に処理されていて(1パスのみが必要)，その処理により各タスク $T$ にはそのレベル $L(T)$ (2.1節参照)が対応づけられていることが仮定されている．スケジューリング・アルゴリズムは次のように記述される．**プロセッサが空いたときはいつも**[†]**，先行タスクの全てが既に割当てられてしまっており，しかもそれ自身はまだ割当てられていないタスクのうち最高のレベルにあるタスク(もしあれば)を割当てる．いくつかのタスクが互角である場合は，そのようなタスクの中から任意の一つを選ぶ．**このアルゴリズムで作られるスケジュールを ***B*-スケジュール**と呼ぶことにしよう．例は図3.3-2(a)にあるが，これはプロセッサ数 $m = 3$ を想定して図3.3-1の木に対する $B$-スケジュールを示したものである．図3.3-2(b)には，同じ時間単位内で実行されるタスク同志を線で囲むことによってこのスケジュールを明示している．自明な方法で $m(>2)$ プロセッサに拡張された $A$-スケジュールが，木に対する正当な $B$-スケジュールを産み出すことに注目すると良い．

$B$-スケジュールが木に対して最適であることを証明する前に，任意のグラフおよび任意のプロセッサ数 $m$ に対して有効な，最短実行時間の下界を導こう．その次に，この下界が実際に(タスクの実行時間が全て等しい)木に対する $B$-スケジュールで達成されることを示そう．

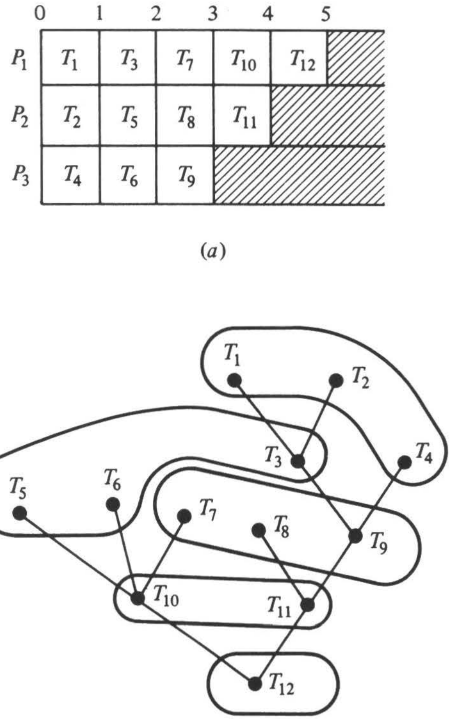
レベル $j$ 以上のタスクの集合を $Q(j)$ で表し，与えられたグラフ $G$ 内の最大長の道を $\mathbf{P} = (T_{i_1}, T_{i_2}, \ldots, T_{i_L})$ とする．任意の $j(0 \le j \le L)$ に対し，レベルが $j$ を越える全てのタスクを $|Q(j+1)|/m$ 単位時間未満で実行することは不可能である．グラフの残りの部分に対しては少なくとも $j$ 単位時間が必要であるから，最適スケジュールの長さ $t_{\min}$ について次式を得る．

$$t_{\min} \ge \max_{0 \le j \le L} \left\{ j + \left\lceil \frac{|Q(j+1)|}{m} \right\rceil \right\} \tag{3.3.1}$$

ここで，記法 $\lceil x \rceil$ は，$x$ 以上の最小の整数を表す．

同様の方法で，与えられたグラフを $t$ 単位時間内で実行するのに必要なプロセッサの最小台数の下界も導くことができることに注意しよう．($L$ はグラフの実行時間に対する自明な下界であるから，$t \ge L$ を仮定しなければならな

† 実際には同時に二つ以上のプロセッサが空くこともあるが，この場合はその添字の順に——$P_{i+2}$ の前に $P_{i+1}$ を，その前に $P_i$ を——スケジュールすることが仮定されていることを思い出そう．




(b)

**図 3.3-2**　図 3.3-1 に対する $B$-スケジュール

い.）この下界は次式で与えられる.

$$m_{\min} \geq \max_{0 \leq j \leq L} \left\lceil \frac{|Q(j+1)|}{t-j} \right\rceil \qquad (t \geq L) \tag{3.3.2}$$

$B$-スケジュールが最適であることの証明を簡単にするため，次の特別な場合を最初に片付けておくと都合が良い.

**補題 3.2**

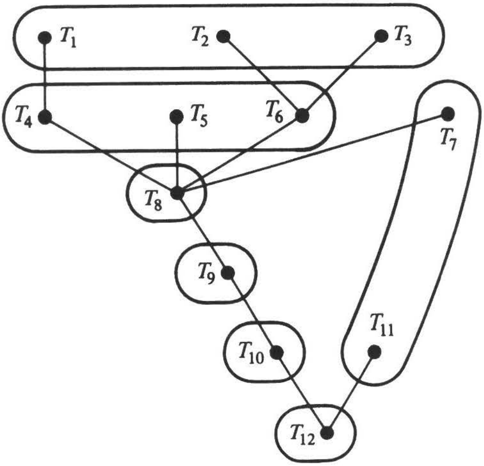
$G$ は，最初に実行可能なタスク（葉である節点）数が $m$ 未満の木であると仮定する．このとき，$G$ の最長の道の長さを $L$ とすると，プロセッサ $m$ 台の場合の $G$に対する$B$-スジケュールは長さ $L$ である.

**証明**　与えられたグラフ $G$ に対する $B$–スケジュールを $\mathcal{S}$ とする．$R_i(0 \leq i \leq t(\mathcal{S})-1)$ を $\mathcal{S}$ の $(i+1)$ 番目の時間間隔で実行可能なタスクの集合とする．$G$ は木であるから，数列 $|R_0|, |R_1|, \ldots, |R_{t(\mathcal{S})-1}|$ は単調非増加である．この重要な知見は，実行可能なタスクを一つ実行すると，次の時間間隔に実行可能となるタスクが生ずるのは高々一つであるという簡単な事実による．したがって，$|R_0| < m$ ということは，全ての $j(1 \leq j < t(\mathcal{S}))$ に対しても $|R_j| < m$ を意味する．この事実を使うと，$R_{j+1}$ 内のタスクは全て，$R_j(0 \leq j < t(\mathcal{S}))$ 内の最高レベルのタスクより低いレベルにあるということを容易に確かめることができる．なぜなら，これが正しくないと仮定すると，$R_j$ 内に先行タスクを持っていないあるタスクが $R_{j+1}$ 内に存在する．$|R_j| < m$ であるので，このタスクは $B$–スケジュールの定義によって $R_j$ に含まれてしまわなければならなくなるからである．したがって，レベル数が $L$ であることから，$t(\mathcal{S}) \leq L$ である．$L$ は $t(\mathcal{S})$ の下界でもあるから，$t(\mathcal{S}) = L$ を得る．　■

**定理 3.2**
$G$ は木であり，$\mathcal{S}_0$ を $G$ に対する $m$ プロセッサの $B$–スケジュールであるとする．このとき，$G$ に対する $m$ プロセッサの全ての正当なスケジュール $\mathcal{S}$ に対し，$t(\mathcal{S}_0) \leq t(\mathcal{S})$．

**証明**　$L$ を $G$ のレベル数とする．定理の結果は $L=1$ に対しては自明であるから，$L \geq 2$ を仮定する．$G$ に対する $m$ プロセッサの，与えられた $B$–スケジュール $\mathcal{S}_0$ の長さを $t$ で表し，このスケジュールで最初の $i(0 \leq i \leq t)$ 単位時間内に実行されるタスクの集合を $N_i$ で表す(ただし，$N_0 = \phi$)．$|N_{p+1} - N_p| < m$ であるような最小の整数を $p$ と定義する．$p=0$ ならば補題 3.2 が適用されてそれで終りであるから，$0 < p < t$ と仮定する．最初の $p$ 単位時間の間はどのプロセッサも空きにはなり得ず，$1 \leq i \leq p$ であるような全ての $i$ に対して，$|N_i|$ は $m$ の整数倍であることに注意しよう．

最初の $(j+1)$ 単位時間の後に実行されるべくしてまだ残っているタスクが木の中で占めている最高のレベルを $v_j(0 \leq j < t)$ と定義する．言い換えると，$Q(v_j+1) \subseteq N_{j+1}$ であるような最小の整数(レベル)を $v_j$ と定義する．図 3.3-3 に $p$ と $v_j$ の定義を示す．後で使うために $j=p$ という特別な場合を考えよう．$Q(v_p+1) \subseteq N_{p+1}$，かつ $N_p \subseteq N_{p+1}$ ではあるが，$N_p$ と $Q(v_p+1)$ とは等しくなり得ない．なぜなら，$N_p = Q(v_p+1)$ であると仮定すれば，最初の $p$ 単位時間でレベル $(v_p+1)$ 以上の全てのタスクを，しかもそれらのみを実行したことになるから，この時点で残っているタスクの最高レベルは $v_p$，す




$m = 3,\ t = 7$
$p = 2, v_p = 4,\ Q(5) = \{1-7\}$
$N_2 = \{1-6\},\ N_3 = \{1-7, 11\},\ N_2 \subset Q(5) \subset N_3$

**図 3.3-3　第 1 の場合の例**

なわち，最初の$(p+1)$単位時間後に残っているタスクの最高レベルとなる．$p$ 時間後，$(p+1)$時間後共に残っているタスクの最高レベルが一致するということは$(p+1)$番目の時間間隔において空いているプロセッサがないことを意味しているが，これは $p$ の定義に矛盾する．

さて，$p$ の定義および補題 3.2 より $(p+1)$ 単位時間後に残っているタスクは，$v_p$ 単位時間で完了することが分る．このことから次式を得る．

$$t = p + 1 + v_p = v_p + \left\lceil \frac{|N_{p+1}|}{m} \right\rceil$$

式(3.3.1)を適用するために，$t$ を $Q(v_p+1)$ で表現したい．次の二つの場合を考える必要がある．

第 1 の場合：$N_p \subset Q(v_p+1)$．［図 3.3-3 の例参照.］定義から $Q(v_p+1) \subseteq N_{p+1}$ であり，この場合は次式が成り立つ†．

$$\left\lceil \frac{|N_{p+1}|}{m} \right\rceil = \left\lceil \frac{|Q(v_p+1)|}{m} \right\rceil$$

---

† **訳注**　$N_p \subset Q(v_p+1)$ かつ $|N_p| = mp$ であるから，$\lceil |Q(v_p+1)|/m \rceil > \lceil |N_p|/m \rceil = p$．一方，$Q(v_p+1) \subseteq N_{p+1}$ であるから，$p+1 = \lceil |N_{p+1}|/m \rceil \geq \lceil Q(v_p+1)/m \rceil$．ゆえに本文の式が成り立つ．

したがって,

$$t = v_p + \left\lceil \frac{|Q(v_p+1)|}{m} \right\rceil \le \max_{0\le j\le L}\left\{ j + \left\lceil \frac{|Q(j+1)|}{m} \right\rceil \right\}$$

ここで式(3.3.1)を適用すると証明は終了する.

第2の場合: $N_p \not\subset Q(v_p+1)$. [図3.3-4の例参照. ここでは $p=3$, $v_p=3$ である.] この場合, $N_p$ は $(v_p+1)$ より低いレベルのタスクを含んでいることに注意しよう†. さらに二つの場合を考える.

**a.** $N_i \subset Q(v_i+1)$ となるような $i(1 \le i \le p)$ が存在しない場合を仮定しよう. このとき, 全ての $i(0 \le i < p)$ に対し, 時間間隔 $(i,\ i+1)$ において実行される最高レベルのタスクの数は $m$ 未満であることが容易に示される. (そのような各 $i$ に対し, $(N_{i+1} - N_i)$ は $(v_i+1)$ よりも低いレベ

---

† **訳注**　この[第2の場合]についてここでは原文に忠実に従ったけれども, 若干の不備もあって些か分り難いので, 参考のため, [第2の場合]の訳者らによる証明を以下に述べる.

**[補題1]** 任意の $i(1 \le i < p)$ に対し, $N_i \subsetneq Q(v_{i-1})$ であるならば, 時刻 $i$ において未実行である最高レベル $v_{i-1}$ のタスク数 $m'$ は $m$ 未満であり, $v_{i-1} = v_i + 1$.

**証明**　背理法による. 時刻 $i$ におけるレベル $v_{i-1}$ の未実行タスクを $T(1)$, $T(2)$, $\cdots$, $T(m')$ とし, $m' \ge m$ であると仮定する. 補題の仮定より, あるタスク $T_w$ が存在して $T_w \in N_i$ かつ $T_w \notin Q(v_{i-1})$. $T_w$ のレベルを $\rho_w(< v_{i-1})$ とし, $T_w$ を実行した時間間隔を $(j,\ j+1)$ とする $(0 \le j < i)$. $T_w$ より高いレベル $v_{i-1}$ の各タスク $T(k)$ $(1 \le k \le m')$ が時間間隔 $(j,\ j+1)$ において実行されなかったのは, 時刻 $j$ において各 $T(k)$ に1個以上の未実行の先行者があったからに他ならない. 各 $T(k)$ に対し, 時刻 $j$ における未実行先行者の最高レベルのタスク(の一つ)を $\underline{T}(k)$ とする. G は木であるから $\underline{T}(1), \underline{T}(2), \cdots, \underline{T}(m')$ は全て異なる. 明らかに各 $\underline{T}(k)$ のレベルは $\rho_w$ よりも高い. したがって, 時間間隔 $(j,\ j+1)$ においては未実行タスク $\underline{T}(1)$, $\underline{T}(2)$, $\cdots$, $\underline{T}(m')$ の中からレベルの高い順に $m$ 個が選ばれて実行されなければならず, $T_w$ が実行されるはずはない. この矛盾は $m' \ge m$ の仮定による. したがって $m' < m$.

また, $p$ の定義から時間間隔 $(i,\ i+1)$ $(i < p)$ において実行されるタスク数は $m$ であるから, $m' < m$ は $v_{i-1} = v_i + 1$ を意味する. ∎

**[補題2]** 任意の $i(1 \le i)$ に対し, $N_i \ne Q(v_{i-1})$.

(証明略)

**[補題3]** $N_i \subset Q(v_{i-1})$, かつ, $N_{i+1} \subsetneq Q(v_i)$ ならば $v_{i+1} = v_i + 1$.

**証明**　$N_i \subset Q(v_{i-1})$ および $v_{i-1} \ne v_i + 1$ を仮定して矛盾を導こう. $v_{i-1} \ne v_i + 1$ であるから $v_{i-1} = v_i$. 時間間隔 $(i,\ i+1)$ においてレベル $v_{i-1}(= v_i)$ よりも低いレベルのタスクを実行するはずはなく, $N_{i+1} \subset Q(v_i)$ となり仮定に反する. ∎

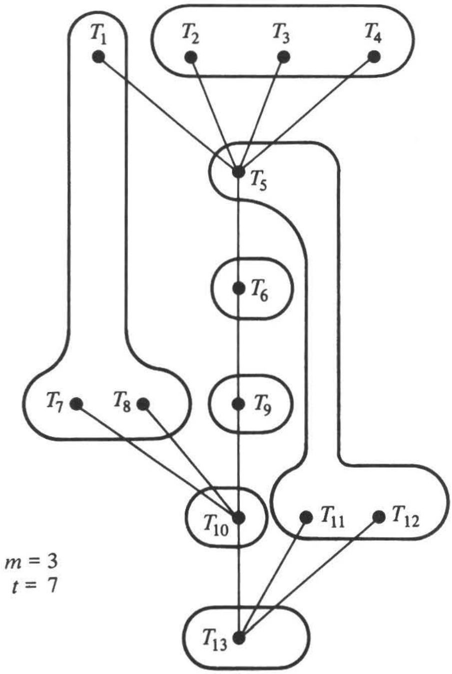
**[補題4]** $v_{p-1} = v_p + 1$.

(証明略)

さて, a. および b. の場合分けの仕方を少し変えて本題の証明に入ろう.

[第2の場合]: $N_p \subsetneq Q(v_p+1)$. この場合, $N_p$ は $v_p+1$ よりも低いレベルのタスクを含んでいることに注意しよう. さらに二つの場合を考える.

**a.** 全ての $i(1 \le i < p)$ に対して $N_i \subsetneq Q(v_{i-1})$ である場合を仮定しよう. $N_1 \subsetneq Q(v_0)$ ゆえ, $G$ の最高レベル $L$ のタスク数は $m$ 未満であることは明らかであり, $v_0 = L-1$. 各 $i(1 \le i < p)$ に補題1を直接適用することにより, 全ての $i(1 \le i < p)$ に対し, $v_{i-1} = v_i + 1$. $p$ の定義と補題3.2より, このことは $i \ge p$ に対しても成り立たなければならないから $t = L$ が得られ, これは明らかに最短である. ↗




$$p = 3,\quad N_3 = \{1-5, 7, 8, 11, 12\}$$
$$N_4 = \{1-8, 11, 12\}$$
$$v_p = 3,\quad Q(4) = \{1-6\}\quad N_p \not\subset Q(v_p+1) \subseteq N_{p+1}$$

**図 3.3-4　第2の場合の例**

---

↘ **b.** $N_i \subseteq Q(v_{i-1})$ であるようなある整数 $i\,(1 \leq i < p)$ が存在すると仮定し，そのような整数のうち最大のものを $s$ とおく．補題2より，$N_s \subset Q(v_{s-1})$．$s$ の最大性から $N_{s+1} \subsetneq Q(v_s)$．補題3から $v_{s-1} = v_s + 1$．$v$ の定義から $Q(v_s + 1) \subseteq N_{s+1}$．以上のことから，次のように書くことができる．

$$t = v_p + ((p+1) - (s+1)) + \left\lceil \frac{|Q(v_s+1)|}{m} \right\rceil$$

次に，$s$ の最大性から全ての $j\,(s+1 \leq j < p)$ に対して $N_j \subsetneq Q(v_{j-1})$ であることから再び補題1を適用でき，全ての $j\,(s+1 \leq j < p)$ に対して $v_{j-1} = v_j + 1$．さらに補題4から，$v_p + (p - s) = v_s$ となる．結局，

$$t = v_s + \left\lceil \frac{|Q(v_s+1)|}{m} \right\rceil \leq \max_{0 \leq j \leq L} \left\{ j + \left\lceil \frac{|Q(j+1)|}{m} \right\rceil \right\}$$

となり，これに式(3.3.1)を適用することによって証明は完結する．　∎

ルのタスクを少なくとも一つは含んでいる.）このことはさらに，全ての $i(0 \leq i < p)$に対し，時間間隔$(i+1,\ i+2)$において実行されるタスクの最高レベルは$(i,\ i+1)$において実行されるタスクの最高レベルのタスクよりも低いことを示している．$p$ の定義と補題3.2より，このことは $i \geq p$ に対しても成り立たなければならないから $t = L$が得られ，これは明らかに最短である.

**b.** 次に，$N_i \subset Q(v_i+1)$となるようなある整数 $i(1 \leq i < p)$が存在すると仮定し，そのような整数のうち最大のものを $s$ とおく．たとえば，図3.3-4 では，$s=1$，$v_s=5$ であり，$N_1=\{T_2,\ T_3,\ T_4\} \subset Q(6)=\{T_1,\ T_2,\ T_3,\ T_4\} \subset N_2=\{T_1 \sim T_4,\ T_7,\ T_8\}$ である．明らかに次のように書くことができる.

$$t = v_p + (p-s) + \left\lceil \frac{|Q(v_s+1)|}{m} \right\rceil$$

しかし，全ての $i(s < i \leq p)$に対して $N_i \not\subset Q(v_i+1)$であることから，全ての $i(s \leq i < p)$に対して $v_i = v_{i+1}+1$ となり，$v_p+(p-s)=v_s$ となる．結局，

$$t = v_s + \left\lceil \frac{|Q(v_s+1)|}{m} \right\rceil \leq \max_{0 \leq j \leq L} \left\{ j + \left\lceil \frac{|Q(j+1)|}{m} \right\rceil \right\}$$

となり，これに式(3.3.1)を適用することによって証明は完結する.　∎

## 3.4 独立なタスクのスケジューリング

任意の実行時間を持つ独立なタスクの集合と任意のプロセッサ数に対するスケジューリング・アルゴリズムを見つけることは，広い範囲にわたる興味も応用もある古典的な順序付けの問題である．この表面的にはなかなかじれったい問題は，基本的に数え上げ法でないどんな解も，受け容れそうにない[20]．しかし，この問題の重要性から，本質的に最適スケジュールを見つける効率の良い技法と共に，発見的手法についての研究や解析に対しても，多大な努力が払われてきた[5]．本節では，独立なタスクに対する横取り権なしのスケジューリングのための，恐らく最も明白かつ単純な発見的手法として得られている興味ある結果に専心する.

独立なタスク・システムに対し，プロセッサが空いたときはいつも，まだ割当てられていないタスクのうち，その処理時間が最長のタスクにそれを割当てるというアルゴリズムによって得られるものを**最長処理時間順**( largest-process-

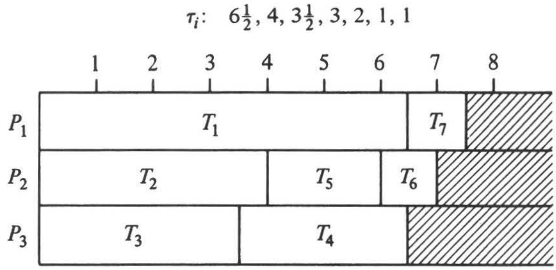



**図 3.4-1** LPT スケジュールの例

ing-time, LPT) スケジューリングと定義しよう．互角のものがあるときは，順序を決めるのにどんな規則を用いても良い．**図 3.4-1** に例を示す．この例の場合，LPT スケジューリングが実際には最適スケジュールを作り出している．**図 3.4-2** の例は，LPT スケジューリングが一般には最適ではないことを示している．

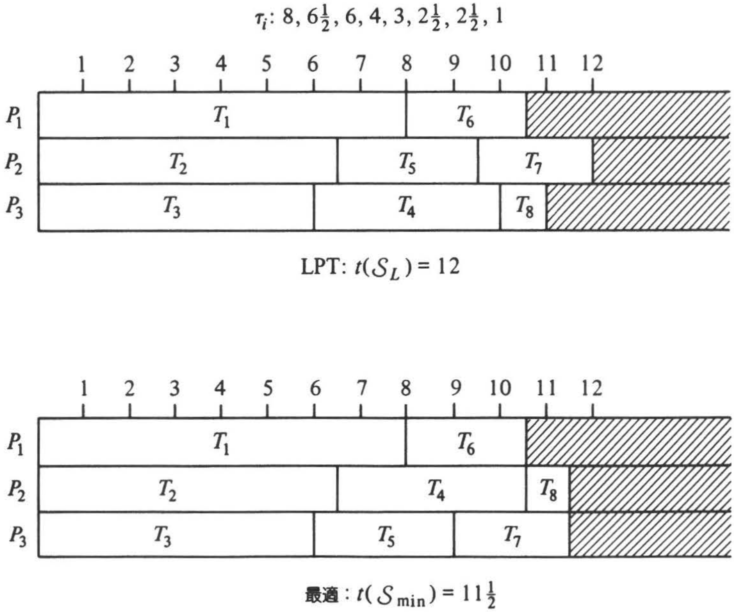
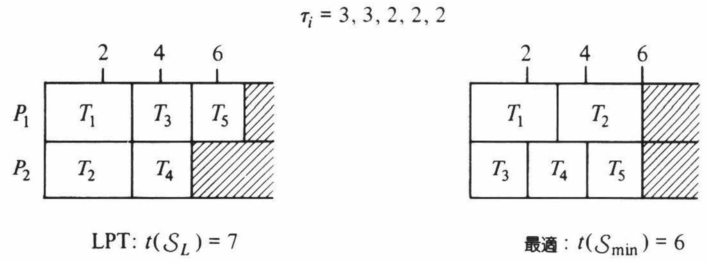


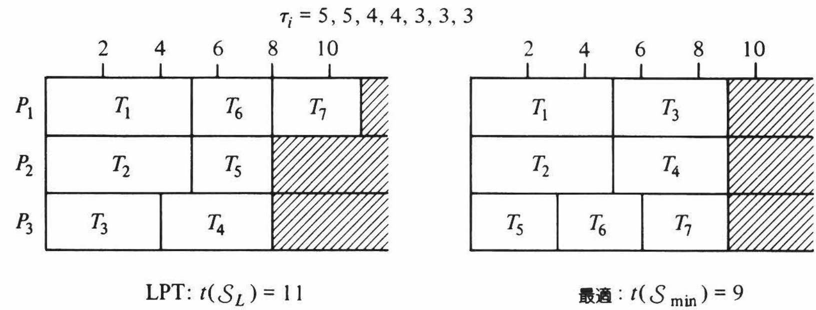

**図 3.4-2** 亜最適な LPT スケジュール

最適スケジューリングに比べて LPT スケジューリングはどの程度良いのかという疑問が生ずるのは自然である．LPT スケジュールにありそうな欠点をはっきりさせるためにいくつかの実験を行うと，プロセッサ数 $m=2$ に対する最も簡単で最悪の例として，図 3.4-3(a)に示したものが得られるのが普通であろう．この仕組みを $m=3$ に拡張したものを図 3.4-3(b)に示す．これを任意のプロセッサ数に拡張すると，結局図 3.4-4 に示した一般例を得る．ただし，ここでは $x$ 以下の最大の整数は $\lfloor x \rfloor$ と表され，タスク $T_i$ の実行時間は $\tau_i = 2m - \lfloor (i+1)/2 \rfloor$ $(i=1, 2, \ldots, 2m)$ および $\tau_{2m+1} = m$ である．図から確かめられるように，$t(\mathcal{S}_L) = 4m-1$ かつ $t(\mathcal{S}_{\min}) = 3m$ であるから，次式は LPT スケジューリングがこの場合の亜最適(suboptimal)である(最適より劣る)ことの程度の最大を表している．

$$\frac{t(\mathcal{S}_L)}{t(\mathcal{S}_{\min})} = \frac{4}{3} - \frac{1}{3m}$$




(a)　$m=2$




(b)　$m=3$

図 3.4-3　LPT に対する最悪の場合

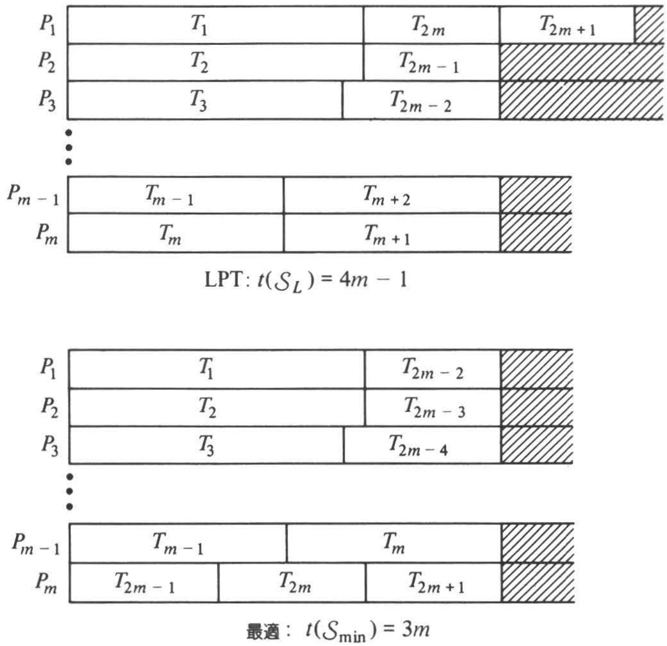



**図3.4-4** 一般の最悪の場合

実際，次の定理に示されているように，図3.4-4はLPTスケジューリングで起り得る最悪の性能を表している [6].

**定理 3.3**

独立なタスクの任意の集合に対し，対応するLPTスケジュールの長さを $t_L$，最適スケジュールの長さを $t_{\min}$ と表す．このとき，次式が成り立つ．

$$\frac{t_L}{t_{\min}} \le \frac{4}{3} - \frac{1}{3m} \tag{3.4.1}$$

**証明** $m=1$ のときは明らかに定理は成り立つから，$m \ge 2$ とする．定理に反して，式(3.4.1)が成り立たないようなタスクの集合 $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ および実行時間 $\{\tau_1, \tau_2, \ldots, \tau_n\}$ があると仮定する．このとき，それよりタスク数の少ない集合で式(3.4.1)が成立しないものは存在しないと仮定してもよい．以下の議論を簡単にするため，タスクはその実行時間が減少するような順番に添字が付けられている(すなわち，$\tau_1 \ge \tau_2 \ge \ldots \ge \tau_n$)ものと仮定し，

互角のもの同志から選ぶときの規則はいつも，この添字が最小のものであると仮定する．これらの仮定の下では，$\mathcal{T}$ に対する LPT スケジュールはいつも $T_1, T_2, \cdots, T_n$ の順にタスクをスケジュールすることになる．

$\mathcal{S}_L$ を $\mathcal{T}$ に対する LPT スケジュールとし，$T_k(k \leq n)$ は $\mathcal{S}_L$ 内で最後の完了時刻($t_L$)を持つタスクの中の一つを表すものと仮定する．タスク $T_i(1 \leq i < k)$ は全て $T_k$ よりも前に割当てられているから，$\{T_1, T_2, \ldots, T_k\}$ に対する LPT スケジュールの長さ $t'_L$ もやはり $t_L$ でなければならないことが直ちに分る．一方，$\{T_1, T_2, \ldots, T_k\}$ のスケジュールに対する最短時間 $t'_{\min}$ は，$t'_{\min} \leq t_{\min}$ を満たしていなければならない．結局，上述の仮定から，

$$\frac{t'_L}{t'_{\min}} \geq \frac{t_L}{t_{\min}} > \frac{4}{3} - \frac{1}{3m}$$

つまり，$k < n$ と仮定すると，タスク数が $n$ より少ない集合が存在してそれが式(3.4.1)を満たさないという矛盾が生ずる．したがって，$T_n \in \mathcal{T}$ は，最後の完了時刻 $t_L$ を持ち得るただ一つのタスクであると仮定してよい．

ここで，（式(3.4.1)を満たさないと仮定した)$\mathcal{T}$ に対する任意の最適スケジュールにおいては，一つのプロセッサに三つ以上のタスクが割当てられることはないということを証明して，問題をより限定しよう．まず，最適スケジュールの長さに対し次のように書くことができる．

$$t_{\min} \geq \frac{1}{m}\sum_{i=1}^{n}\tau_i \tag{3.4.2}$$

式(3.3.1)は，タスクが独立でなく，かつ同じ実行時間を持っている場合について同じことを表現したものであることに注意しよう．さて，$\mathcal{S}_L$ における $T_n$ の開始時刻を $t_n$ で表す．$T_n$ が実行を始めるまではどのプロセッサも遊休期間を持ち得ないから，次式を得る．

$$t_n \leq \frac{1}{m}\sum_{i=1}^{n-1}\tau_i \tag{3.4.3}$$

ゆえに，

$$\frac{t_L}{t_{\min}} = \frac{t_n + \tau_n}{t_{\min}} \leq \frac{\tau_n}{t_{\min}} + \frac{1}{mt_{\min}}\sum_{i=1}^{n-1}\tau_i$$

すなわち，

$$\frac{t_L}{t_{\min}} \le \frac{(m-1)\tau_n}{mt_{\min}} + \frac{1}{mt_{\min}} \sum_{i=1}^{n} \tau_i$$

式(3.4.2)を使うと，

$$\frac{t_L}{t_{\min}} \le \frac{(m-1)\tau_n}{mt_{\min}} + 1$$

式(3.4.1)が成り立たないことが仮定されているから，

$$\frac{(m-1)\tau_n}{mt_{\min}} + 1 > \frac{4}{3} - \frac{1}{3m}$$

これから次式を得る．

$$t_{\min} < 3\tau_n$$

したがって，$T_n$ の実行時間が最短であることから，もし式(3.4.1)が成立しないのであれば，$\mathcal{T}$ に対する最適スケジュールではどのプロセッサも三つ以上のタスクを実行し得ないことになる．

長さ $t_0$ の上記のような最適スケジュールの一つを $\mathcal{S}_0$ としたとき，$\mathcal{S}_0$ より長くはなく，かつ LPT スケジュールと同型なスケジュール $\mathcal{S}^*$(ここで，二つのスケジュールが同型であるとは，一方のスケジュールのプロセッサ間および実行時間が等しいタスク間での名前のつけ替えによって他方と同じにできることをいう)に $\mathcal{S}_0$ を変換できることを次に示そう．その結果，$t^*_0 = t_0$ となることから式(3.4.1)が成り立たないという $\mathcal{T}$ の仮定との間に矛盾が生じ，定理が成立することになる．

まず，$\mathcal{S}_0$ に適用したときにスケジュール長を増加させない変換を 3 種定義することから始めよう．第 1 型と呼ぶ最初の変換は **図 3.4-5** に示されている．

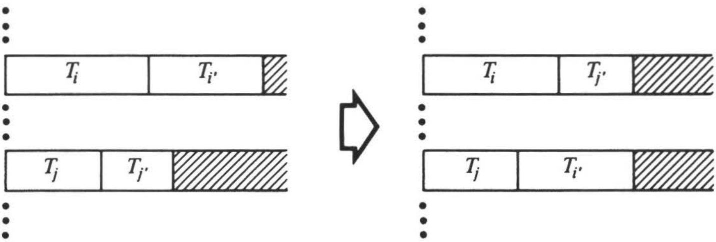



**図 3.4-5　第 1 型変換**

$\tau_i > \tau_j$ かつ $\tau_{i'} > \tau_{j'}$ ならば $T_{i'}$ と $T_{j'}$ の割当てを図のように入れ替えても全体のスケジュール長は長くはならないということが容易に分る．第2型変換を図3.4-6に示す．この図で，$\tau_i > \tau_j$ と仮定すると，変換後のスケジュールは明らかにもとのスケジュールより長くはなっていない．さらに，第1型および第2型が持つ次の性質を容易に確かめることができる．プロセッサ $P_i$ が $\mathcal{S}_0$ および変換後のスケジュール $\mathcal{S}'_0$ で，初めて空きになる時刻をそれぞれ $I_i$ および $I'_i$ で表すと，次式が成り立つ．

$$\sum_{1 \le i < j \le m} |I'_i - I'_j| < \sum_{1 \le i < j \le m} |I_i - I_j|$$

第1型および第2型の変換は有限回だけ $\mathcal{S}_0$ に適用できることが結論としていえる[†]．

最後に，図3.4-7に示された第3型変換は，二つのタスクが割当てられているプロセッサ上のそれらのタスクについて，減少する列に並べかえるだけである．明らかにこの変換はスケジュールの長さには何の影響も与えない．しかもタスク数もプロセッサ数も有限だから，$\mathcal{S}_0$ について行うことのできる一連の第3型変換の回数も有限である．

この3変換の定義から，これらの変換を適用し尽し，プロセッサの順序を適当に入れ替えると，図3.4-8に示したスケジュール $\mathcal{S}^*_0$ を得ることになる．プロセッサ割当てのどの対をとっても，第1, 2, 3型変換を適用できないので，次式が成り立つ．

$$\tau_{k_1} \ge \tau_{k_2} \ge \cdots \ge \tau_{k_r} \ge \tau_{i_1} \ge \cdots \ge \tau_{i_s} \ge \tau_{i'_s} \ge \cdots \ge \tau_{i'_1}$$

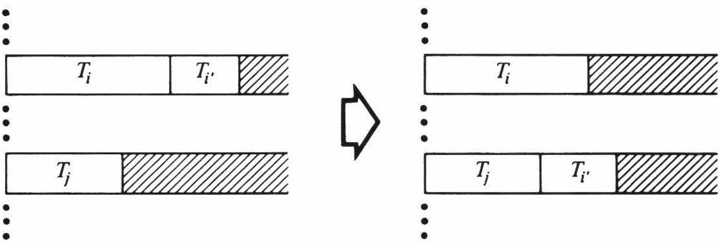



図3.4-6　第2型変換

† **訳注**　第1型および第2型の変換が有限回で終らないと仮定する．プロセッサ数，タスク数ともに有限，したがって，可能なスケジュール数も有限であることから，あるスケジュール $\hat{\mathcal{S}}$ があって，これらの変換の過程に2度以上現われなければならない．これは $\sum|I_i - I_j|$ が変換の度毎に単調に減少していくことと矛盾する．

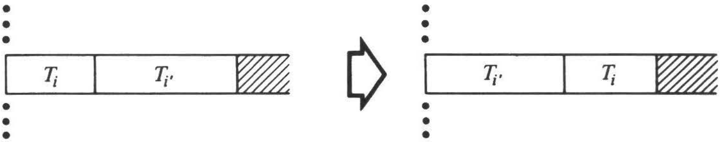



図 3.4-7　第 3 型変換

さらに，これらの変換はスケジュール長を長くせず，$t_0$ が最適であることから，$t_0^* = t_0$ である．次に，一見して LPT スケジュールである $\mathcal{S}^*_0$ が，本当にそうであることを証明しよう．それで証明は完結する．図 3.4-8 を良く調べてみると，$\mathcal{S}^*_0$ が LPT スケジュールと異なり得るのは，**図 3.4-9** 示したように二つのプロセッサへの割当て方が，ある $k < p$ に対して $\tau_{i_k} > \tau_{i_p} + \tau_{i'_p}$ である場合だけであることが分る．この場合，$T_{i'_p}$ の終了後に $P_v$ に $T_{i'_k}$ を割当てると明らかに LPT スケジュールに合致する．しかし，このことは，あるプロセッサに三つ以上のタスクが割当てられる最適スケジュールが存在することになる．この矛盾は，（図 3.4-9 のような場合が存在しないことを示しているから）$\mathcal{S}^*_0$ は正に LPT スケジュールであるという事実を証明しており，さらにこの事実は式(3.4.1)を満たさないタスクの集合が存在するという仮定に反する．■

## 3.5　リスト・スケジューリング

単純さという重要性，あるいは，恐らくタスクの実行時間は事前には分らな

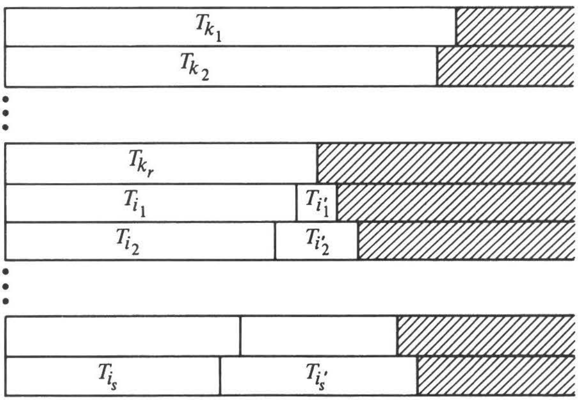



図 3.4-8　スケジュール $\mathcal{S}^*_0$

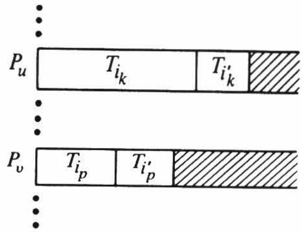
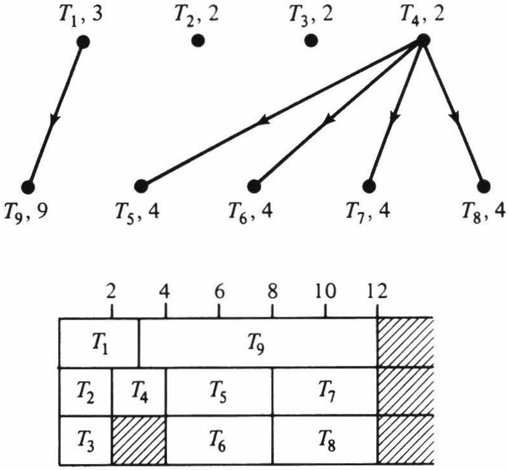



図 3.4-9　$\mathcal{S}^*_0$ についての例

いであろうという理由から，与えられたタスクのリストとそれらのタスクの順序づけを制約する半順序だけを用いるスケジューラを考えることは値打ちがある．プロセッサが空くといつも，スケジューラは単に一定方向にこのリストを走査し，先行タスクが全て終了している未実行タスクの中で最初に見つかったものをそのプロセッサに割当てる．このリスト・スケジューラ(list scheduler)に付託されるタスクのリストは，通常，何らかの外部の優先順位機構に従って並べられていると仮定する．図 3.5-1 に一例が示されているが，ここで $\mathcal{L}$ は，いわゆるタスクの**優先順位リスト**(priority list)を表している．プロセッサが空いたときはいつでも，$\mathcal{L}$ は最初の未実行かつ準備済みのタスクを探すために，左から右へ走査されるものと仮定する．




$\mathcal{L} = (T_1, T_2, \ldots, T_9)$

図 3.5-1　変則の例

これまで議論してきたスケジューリング・アルゴリズムはどれもリスト・スケジューリングの形で表現できる．たとえば，$A$–スケジュールを生成するアルゴリズムは，対応するラベルが減少列になるようにタスクを並べた優先順位リストを持つリスト・スケジューラとして示される．前節で議論したLPTスケジューリング・アルゴリズムは，優先順位リストが実行時間順に並べられたものであるときのリスト・スケジューリングである．本節の目的は，リスト・スケジューリングの持つある種の変則的な性質を明らかにし，それに対応するスケジュール長の増加分の上界を見つけることである．それに続いて，これらの変則を克服する手法を簡単に概観する．

リスト・スケジュールの長さは，プロセッサ数，タスクの実行時間，先行関係の制約を表す半順序，および使われる優先順位リストに依存する．リスト・スケジューリングの変則的な振舞とは，プロセッサ数が増えるとか，実行時間が短くなるとか，ある先行制約を取除くとか，別の優先順位リストを用いるとかしたときにスケジュール長が**増加**するような事例のことをいう．Graham [7] は，これらの四つの変則のどれもが現れる一つの例を与えたが，この例を図 3.5–1 に示す．

まず最初に，図 3.5–1 の他のパラメータを一定に保ち，プロセッサ数を 3 台から 4 台に変更したと仮定しよう．すると，図 3.5–2 に示したように，スケジュール長が 12 から 15 へ増加するという直観に反した結果を得る．次に，新しい実行時間が $\tau'_i = \tau_i - 1$ で与えられるように実行時間を定義し直そう．図 3.5–3 は，スケジュール長が実際に 12 から 13 に延びたことを示している．

図 3.5–4 には，半順序関係を緩めたとき，すなわち，ある先行関係の制約を取除いたときに生ずる変則が示されている．図示されているとおり，$T_5$ および $T_6$ が $T_4$ と独立になるように制約を取除くと，スケジュール長は 12 から 16 に延びる．最後に，優先順位リストを $\mathcal{L}$ から $\mathcal{L}' = (T_1, T_2, T_4, T_5, T_6, T_3, T_9, T_7, T_8)$ に変更すると仮定する．このときも図 3.5–5 に示したよう

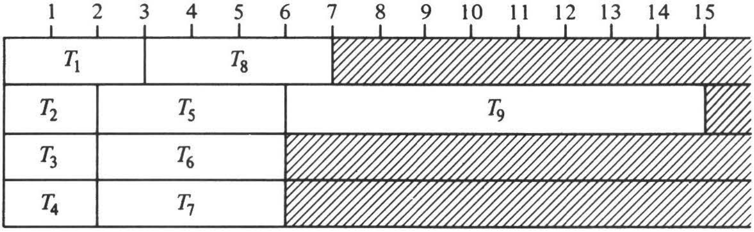



**図 3.5–2** $m$ を 4 に増やした場合

$\tau_i'$: 2, 1, 1, 1, 3, 3, 3, 8

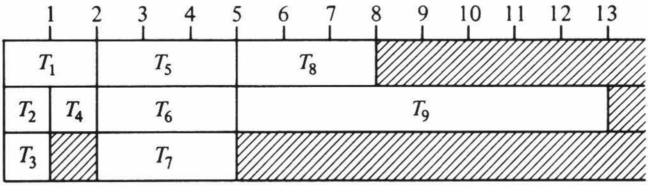



**図 3.5-3**　実行時間を減らした場合

に，スケジュール長は長くなる．

さて，上に述べたような変則がスケジュール長に及ぼす効果の一般的な上界を導いてみよう．しかし，その前に，多重プロセッサのリスト・スケジュールの一般的な性質を明らかにしておくと便利であろう．特に，任意のタスク・システム $C$ および対応するスケジュール $\mathcal{S}$ に対し，あるタスクの連鎖が $C$ 内に存在し，$\mathcal{S}$ でプロセッサが空いているときはいつもその連鎖内のタスクの一つが必ず実行中であるということを示したい．

$\mathcal{S}$ をグラフ $G$ に対する一つの正当なスケジュールとし，$t$ を $\mathcal{S}$ の長さとする．$\mathcal{S}$ で完了時刻が最後である任意のタスク $T_{i_1}$ を考え，$\mathcal{S}$ における $T_{i_1}$ の開始時刻を $t_{i_1}$ とする．まず，$t_{i_1}$ では少なくとも一つのプロセッサが遊休期間中であると仮定する．すなわちあるプロセッサがあって，それは区間$(t_{i_1}-\epsilon, t_{i_1}+\epsilon)$では空いているといえるような $\epsilon>0$ が存在すると仮定する．空いているプロセッサの一つに $T_{i_1}$ がもっと早くに割当てられなかったのは，$t_{i_1}$ で終了

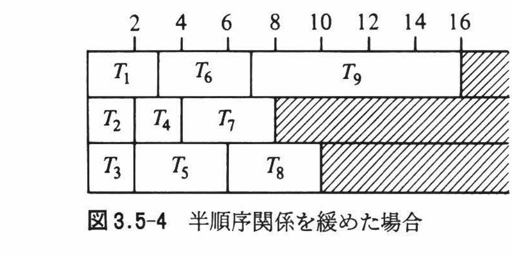
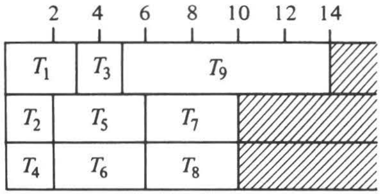



**図 3.5-4**　半順序関係を緩めた場合




$\mathcal{L}' = (T_1, T_2, T_4, T_5, T_6, T_3, T_9, T_7, T_8)$

図 3.5-5 優先順位リストを変更した場合

し，かつ $T_{i_1}$ に先行しなければならないような，すなわち，$T_{i_2} \lessdot T_{i_1}$ であるようなあるタスク $T_{i_2}$ が存在するからである．

今度は，そうではなく，他のどれかのプロセッサの遊休期間の内部(境界は含まない)に $t_{i_1}$ が含まれていないと仮定しよう．プロセッサの遊休期間のうち，その終了時刻が $t_{i_1}$ よりも前で，かつ最後のものを$(t', t'+\epsilon)$ $(t' \geq 0)$で表す．ここでもまた，この時間間隔内で実行中のタスク $T_{i_2} \lessdot T_{i_1}$ が存在していなければならないといえる．もしそうでなければ，$T_{i_1}$ またはその先行タスクの一つがこの遊休期間に割当て得たはずだからである．

つぎに，$T_{i_2}$ に焦点を当てると，同様の議論で，$t_{i_2}$ 以前 ($t_{i_2}$ を含む) に遊休期間がもしあれば，この遊休期間に実行中のあるタスク $T_3 \lessdot T_2$ がやはり存在しなければならないといえる．帰納的に，次の補題にまとめられているように望みのタスクの連鎖を得ることができる．

**補題 3.3**

$\mathcal{S}$を，与えられたタスク・システム $C = (\mathcal{T}, \lessdot)$に対する任意のスケジュールとする．このとき，あるタスクの連鎖

$$T_{i_1} \lessdot T_{i_2} \lessdot \cdots \lessdot T_{i_s} \qquad (T_{i_j} \in \mathcal{T}, 1 \leq j \leq s)$$

が存在し，期間$(0, t(\mathcal{S}))$内にあるプロセッサの遊休期間が含まれているときは必ず他のプロセッサの中の一つがこの連鎖内のタスクを実行している．

この補題を用いると，リスト・スケジュールの変則が及ぼす影響についての次の上界を容易に証明できる．

**定理 3.4**

与えられたタスクの集合を $\mathcal{T} = \{T_1, T_2, \cdots, T_n\}$とし，$m, \tau_1, \tau_2, \ldots, \tau_n$, $\mathcal{L}$ および $\lessdot$ をそれぞれ適当に選択されたプロセッサ数，実行時間，優先順位

リストおよび半順序とする．$m'$, $\tau'_1$, $\tau'_2$, $\cdots$, $\tau'_n$, $\mathcal{L}'$ および $<'$ は，制約‘$\tau'_i \le \tau_i (1 \le i \le n)$，かつ，半順序 $<'$ は半順序 $<$ に含まれている．すなわち $<'$ の各先行関係は $<$ に含まれている’を満たすような別の選択であるとする．$\mathcal{T}$ に対するリスト・スケジュールをこれらの二つの選択に対応して $\mathcal{S}$ および $\mathcal{S}'$ と表し，それぞれのスケジュール長を $t$ および $t'$ と表すとする．このとき，

$$\frac{t'}{t} \le 1 + \frac{m-1}{m'} \tag{3.5.1}$$

**証明** $\mathcal{S}$ の区間$(0, t)$内のプロセッサ $P_i$ の遊休時間の合計を $\phi_i$ と表す．同様に $\mathcal{S}'$ に対して $\phi_i'$ を定義する．$\mathcal{S}'$ に補題3.3を適用すると，あるタスクの連鎖

$$T_{i_1} <' T_{i_2} <' \cdots <' T_{i_s} \tag{3.5.2}$$

が存在して，$\mathcal{S}'$ でどれかのプロセッサが空いているときはいつも，この連鎖内のあるタスクが実行中であるという性質がある．遊休時間の合計は，$T_{i_k}$ の一つが実行中に残りの$(m'-1)$台のプロセッサ全てが空いていると仮定したときに得られる遊休時間の合計を越えることはあり得ないという簡単な理由から，直ちに次式を得る．

$$\sum_{i=1}^{m'} \phi'_i \le (m'-1) \sum_{k=1}^{s} \tau'_{i_k} \tag{3.5.3}$$

さて，半順序 $<$ は半順序 $<'$ を含んでいることから，式(3.5.2)は

$$T_{i_1} < T_{i_2} < \cdots < T_{i_s}$$

であることをも意味しており，次式を得る．

$$t \ge \sum_{k=1}^{s} \tau_{i_k} \ge \sum_{k=1}^{s} \tau'_{i_k} \tag{3.5.4}$$

次の関係

$$t' = \frac{1}{m'}\left\{\sum_{j=1}^{n} \tau'_j + \sum_{i=1}^{m'} \phi'_i\right\}$$

から，式(3.5.3)および式(3.5.4)の結果として次の不等式を得る．

$$t' \le \frac{1}{m'}\Big[mt + (m'-1)t\Big]$$

この不等式から式(3.5.1)は直接導かれる．■

リスト・スケジューリングは，完了時刻に**納期**(dead line)の制限が課せられているという厳しい実時間の環境下において計算をスケジュールするという問題で特別に強調されてきた(問題3.11，3.12，および3.13参照)．タスクの実行時間に対して最大値だけしか分らないときは，実行時間が減ることから生ずる変則の現れ方はいくらか重大である．したがって，最大実行時間をもとに納期に間に合うようにスケジュールを立案するとき，実際の実行時間がより短くなってもその結果得られるスケジュール長が許容できる納期を越えないということを保証されていることが望ましい．

この問題の直接的な解は，もとのグラフに次のような方法で適当な先行関係を追加することによって得られる．もとのグラフで互いに独立なタスク $T_i$ と $T_j$ を考える．各タスクに対して最大の実行時間を想定して，所与の優先順位リストに従ったときに得られるリスト・スケジュールによれば，ある同一のプロセッサ上で $T_i$ が $T_j$ に先行することになるものとしよう．そのような $G$ 内のタスクの対の全てに対して，先行関係 $T_i \lessdot T_j$ を $G$ の半順序につけ加えることによって新しいグラフ $G'$ を作る．同じ優先順位リストを用いるならば，グラフ $G'$ は実行時間が減ることから生ずる変則から免れることは容易に示される．この問題に対して，これとは別の類似した解が [8] に述べられている．その解は，より少なくかつ十分な数の先行関係を $G$ に追加する方法であることから，より高い融通性が得られている．

式(3.5.1)が，得られる最良の上界であることを示す例を与えることができる．実際，この上界には，パラメータ $\mathcal{L}$，$\lessdot$，$m$，および$\{\tau_i\}$の**どの一つ**を変えても達することができる．そのような一つの例を図 3.5-6 に示すが，ここで変えたパラメータは優先順位リストだけである．これを見て分るように，二つのリスト $\mathcal{L}$ および $\mathcal{L}'$ に対応するスケジュール長は，$t'/t = 2 - 1/m$ になり，$m = m'$ であるときの式(3.5.1)の値となっている．他の例は問題3.10に挙げてある．

## 3.6　横取り権のあるスケジューリングとプロセッサの共有

横取り権の働きを導入して，タスクが一たび開始すると完了まで実行し続けなければならないという制約を取除くと，一般にスケジュール時間を改善し得るということが分る．簡単な例が図 3.6-1 に示されているが，実行時間が単位時間である三つの独立タスクに対する最適スケジュールに対して 25% の節約

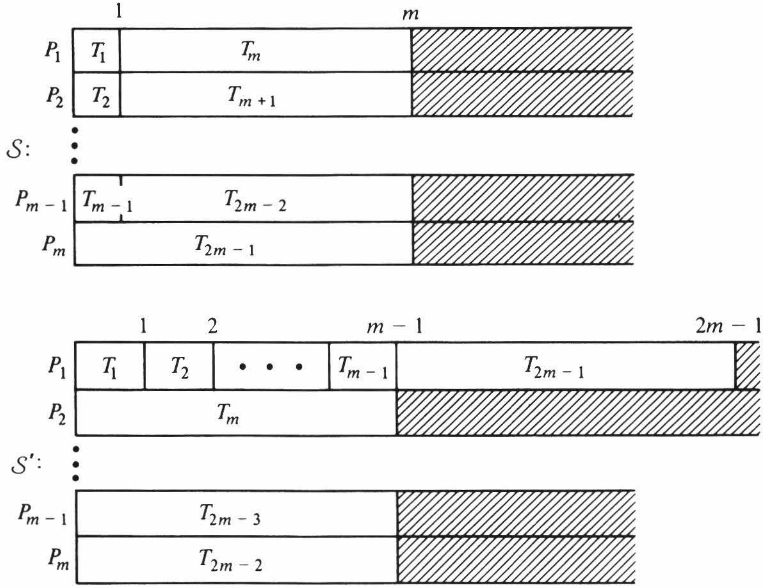
$C = (\mathcal{T}, \lessdot)$　プロセッサ$m$台

$\mathcal{T} = \{T_1, T_2, \ldots, T_{2m-1}\}$, $\lessdot = \phi$ (独立タスク)

$\tau_i = 1 \ (1 \leqslant i \leqslant m-1)$, $\tau_i = m-1 \ (m \leqslant i \leqslant 2m-2)$, $\tau_{2m-1} = m$

$\mathcal{L}$: $(T_1, T_2, \ldots, T_{m-1}, T_{2m-1}, T_m, T_{m+1}, \ldots, T_{2m-2})$

$\mathcal{L}'$: $(T_1, T_m, T_{m+1}, \ldots, T_{2m-2}, T_2, T_3, \ldots, T_{m-1}, T_{2m-1})$




$t(\mathcal{S}')/t(\mathcal{S}) = 2 - 1/m$

**図3.5-6**　優先度リストのみを変えた場合

が達成されている．もちろん，図**3.6-1**では，タスクの横取りに関して目に見える程の費用はかからないということを仮定している．これは，計算機への応用に際しては，一般には当を得ていないであろう．なぜなら，横取りは主記憶からタスクを追出し，その後主記憶に再び搭載する(reload)ことを意味しているからである．しかし，入出力操作が十分に並行動作できない場合は，持込まれる時間遅れは，横取りを考えることをしばしば最初から断念させる程大きい．とにかく，理論上の結果が得られている横取り権ありスケジュールのモデルでは，横取りの費用は無視できることを仮定することが必要である．

独立なタスクの集合に対して，最適な横取り権ありスケジュールの結果はしばらく前から知られていた [9]．

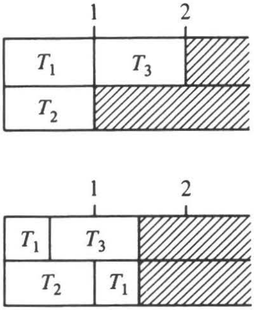



**図 3.6-1**　横取り権による改善

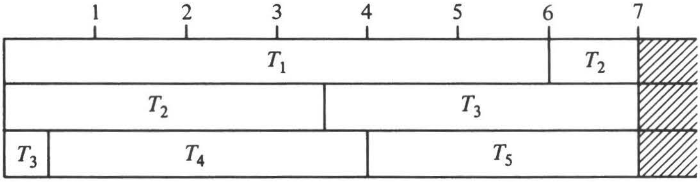
**定理 3.5**

$\mathcal{T}=\{T_1, T_2, \ldots, T_n\}$を独立なタスクの集合とし，$T_i(1 \leq i \leq n)$の実行時間を $\tau_i$ とする．$\mathcal{T}$ に対するプロセッサ $m$ 台の最適な横取り権ありスケジュールの長さを $t_{\min}$ とする．このとき，

$$t_{\min}=\max\left\{\max_{1\leq i\leq n}\{\tau_i\},\ \frac{1}{m}\sum_{i=1}^{n}\tau_i\right\} \tag{3.6.1}$$

**証明**　実行時間が最大のタスクを実行するのに要する時間よりも短い時間で終るスケジュールはあり得ないし，スケジュールは全プロセッサを働かせ続ける以上に効率的ではあり得ないから，式(3.6.1)は $t_{\min}$ についての一つの下界に違いないということは明らかである．この下界が実際に達成できることを見るために，次のようなプロセッサ $m$ 台の $\mathcal{T}$ のスケジュールを工夫する(構造は図 3.6-2 に示されている)．$t=(1/m)\sum_{i=1}^{n}\tau_i$ と置く．

**場合 1．** $t>\max_{1\leq i\leq n}\{\tau_i\}$であると仮定する．$j$ を $t<\sum_{i=1}^{j+1}\tau_i$ であるような最小の整数とするとき，タスクの系列 $T_1, T_2, \ldots, T_j(j\geq 1)$を $P_1$ に割当てる．




$\tau_i: 6, 4\frac{1}{2}, 4, 3\frac{1}{2}, 3$

**図 3.6-2**　五つの独立タスクの場合の例

この段階で $T_{j+1}$ を区間 $(\sum_{i=1}^{j}\tau_i,\ t)$ だけ $P_1$ に割当て，$t'=\sum_{i=1}^{j+1}\tau_i-t$ と置いたとき，区間 $(0,\ t')$ だけ $P_2$ に割当てる．すなわち，$P_1$ で実行されない $T_{j+1}$ の部分を $P_2$ で実行する．$t>\tau_{j+1}$ であるから，このようにしても $(0,\ t)$ 間の任意に与えられた時点で $T_{j+1}$ が $P_1$ と $P_2$ の両方に割当てられているようなことにはなり得ないことが分る．次に，$P_2$ にタスク $T_{j+2},\ T_{j+3},\ \ldots,\ T_{j+k}$ を次々と割当てていき，タスクの割当て($T_{j+k+1}$ に対する)が，もう一度区間 $(0,\ t)$ を越えるまで行う．この段階で $T_{j+k+1}$ は $P_2$ の残りの時間を割当てられ，$T_{j+k+1}$ の残りは $P_3$ の最初のタスクとして割当てられる．$T_n$ への割当てが済むまでこのように進めていく．$t=(1/m)\sum_{i=1}^{n}\tau_i$ かつどのプロセッサにも $(0,\ t)$ の間に空き時間がないことから，$T_n$ は時刻 $t$ に $P_m$ 上で完了することが分る．また，全ての $i(1\leq i\leq n)$ に対して $\tau_i<t$ であるという事実から，任意に与えられたタスクに対して同時に二つ以上のプロセッサが割当てられているような時刻は存在しないということも分る．

**場合 2.** $\tau_{\max}=\max_{1\leq i\leq n}\{\tau_i\}\geq t$ であると仮定すると，少なくとも一つのタスクは，その全実行時間が一つのプロセッサに割当てられなければならない．上述のスケジュールの構成法から，$\mathcal{T}$ を区間 $(0,\ \tau_{\max})$ 内に割当て得ることは明らかである．∎

横取り権を認めるスケジューリングに密接に関連したスケジューリングの規律として，**プロセッサの共有**(processor sharing)がある．この規律では，$m$ 台のプロセッサは別々の資源単位というよりもむしろ，ある一定量の計算能力から成っているものと考えられる．この計算能力は，零からプロセッサ1台分までの間の(一般にはタスクによって異なる)任意の量がタスクに与えられるものと仮定する．これまでと同じく，タスクは1台のプロセッサで達成できる速度より速く実行されることは決してない．タスク毎に計算能力のある一定の(実行中に不変な)量 $\beta(0\leq\beta\leq 1)$ が割当てられる†．その際，タスクの実行時間は $\dfrac{1}{\beta}$ 倍に増加すると仮定する．たとえば，実行時間 $\tau$ のタスク $T$ に対して $\beta=\dfrac{1}{2}$ ならば，$T$ の実行時間は完了するまでに $2\tau$ 時間単位を要する．1例が**図3.6-3**に示されている．プロセッサの共有ありスケジューリングに対するガント図表現は，自明の方法で拡張すればよい．すなわち，割当てられた $\beta$ を表すためには，単に垂直方向に分割された区間の幅を利用すればよい．横取り権もプロセッサ共有もない最良スケジュールは4時間単位必要であるのに対

† **訳注**　計算能力をタスク間に等分に分配することは仮定していない．

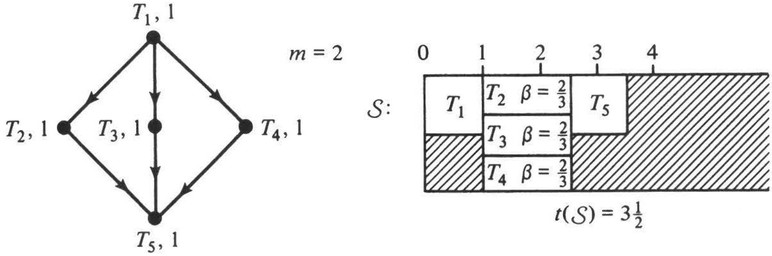



**図 3.6-3** プロセッサの共有

し，プロセッサ共有スケジュールは，$3\frac{1}{2}$時間単位だけでよいことに注意しよう．上の定義により，タスクに割当てられた計算能力の割合は，そのタスクの実行中には変更されないということを強調しておく．

プロセッサの共有を物理的にどのように実現するかは思い浮かべ難いかも知れないが，この概念は二つの理由で有用である．第一に，横取り権ありの最適スケジューリング・アルゴリズムの記述が大いに簡単になるし，第二に，タイムシェアリング・システムにおけるスケジューリング・アルゴリズムの有用で比較的単純なモデルにもなり得る(第4章)からである．

タスクに割当てられた計算能力の量をタスクが完了する前に変えても良いとき(ある区間はタスクが全く実行されないという場合も含む)，これを**一般スケジューリングの規律**(general scheduling discipline)という．与えられたタスクに対する $\beta$ の値をそのタスクが完了する前に変えても良いとしたとき，あからさまな横取りの能力を効果的に導入したことになるのは明らかであろう．当然，あらゆる横取り権ありスケジュールは一般スケジュールであるということになる．最適な一般スケジュールの例を**図 3.6-4** に示す．タスク $T_3$ と $T_5$ の二つは，その実行中を通して割当てられた計算能力が一定であったわけでは

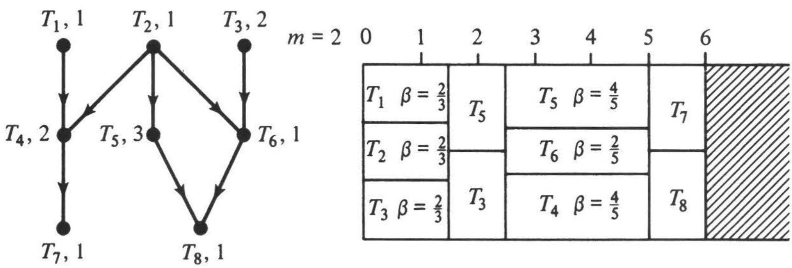



**図 3.6-4** 一般スケジューリング

ないことに注意しよう．

プロセッサの共有，横取り権あり，および一般スケジュールについてその有効性の相互比較を明確にすることは大いに興味のあることである．このために，次の術語を導入する．与えられたプロセッサ数に対し，1)与えられたグラフについて規律 $D_2$ に従うどのようなスケジュールに対しても規律 $D_1$ に従う高々同じ長さのスケジュールが存在し，かつ，2)あるグラフが存在して，$D_1$ に従うその最適スケジュールが，$D_2$ に従うどのようなスケジュールよりも短くなるとき，しかもそのときにのみ $D_1$ は $D_2$ **よりも有効である**(more effective)という．全てのグラフに対し，$D_1$ と $D_2$ の規律に従う最適スケジュールが同じ長さになるとき，$D_1$ と $D_2$ は**同等に有効である**(equally effective)という．

図3.6–3は，プロセッサ共有ありの(したがって一般の)スケジューリングは横取り権のないスケジューリングよりも有効であることを示している．また，図3.6–1は，横取り権ありスケジューリングは横取り権なしよりも有効であることを示している．次の結果は，プロセッサ共有，横取り権あり，および一般スケジューリングを比較したものである[10]．

**定理 3.6**

1. 一般スケジューリング規律と横取り権のある規律は同等に有効である．
2. 横取り権のある(したがって一般の)スケジューリング規律は，プロセッサ共有の規律よりも有効である．

**証明**　1. 与えられたグラフ $G$ に対するプロセッサ $m$ 台の任意の一般スケジュールが，$G$ に対する同じ長さの横取り権ありスケジュールに変換できることを示すことができれば十分である．$\mathcal{S}$ を $G$ に対する一般スケジュールとし，$\mathcal{S}$ でタスクの割当てが変化する時刻の系列を $t_1 < t_2 < \cdots < t_k$ とする．たとえば，図3.6–4で，$k=4$, $t_1 = 1\frac{1}{2}$, $t_2 = 2\frac{1}{2}$, $t_3 = 5$, $t_4 = 6$ である．区間$(0, t_1)$ $(t_1, t_2)\cdots(t_{k-1}, t_k)$には，同じ区間に割当てられている任意の二つのタスクは独立でなければならないという性質がある．なぜなら，もしそうでなければ，そのタスクのうちの一つはその区間内で完了しなければならず，このことは系列 $t_1 \ldots t_k$ の定義に矛盾する．

区間$(t_i, t_{i+1})$を考え，$T_{i_1}, T_{i_2}, \ldots, T_{i_s}$ がこの区間内でそれぞれ計算能力割当て量 $\alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_s}$ で実行されているとする．つまり，タスク $T_{i_j}$$(1 \le j \le s)$は$(t_i, t_{i+1})$内で $\alpha_{i_j}(t_{i+1} - t_i)$ 単位の実行時間を得ることになる．さて，実行時間 $\alpha_{i_j}(t_{i+1} - t_i)$を使ってこの区間に対する同じ長さの横取り権ありのス

ケジュールを生成するために，定理3.5の構成法を単純に適用する．これを全ての区間$(t_i, t_{i+1})$に対して行えば，もとの一般スケジュールは，長さが同じでありどの先行関係の制約にも違反しない$G$の横取り権ありケスジュールに変換される．

2．上述の1.は，プロセッサの共有は横取り権ありスケジュールよりも有効にはなり得ないことを示しているから，最適の横取り権ありスケジュールが最適のプロセッサ共有スケジュールよりも短い例を一つ与えれば十分である．そのようなグラフの例とそれに対する最適の横取り権ありスケジュールを図3.6-5に示す．最適のプロセッサ共有スケジュールは，この最適横取り権ありスケジュールを越える長さになることを確かめるのは難しくない． ∎

次に示す結果では，横取り権のないスケジューリングと比べたとき，横取り権のある(したがって一般の)スケジューリングに改めることによって得られる最大の改善度の上界を示す[3, 10]．しかし，まず最初に，道の長さとレベルの定義をつぎのように拡張しておこう．$T_{i_1}, T_{i_2}, \ldots, T_{i_s}$をある与えられた連鎖(または道)内のタスクとする．このとき，この連鎖の道の長さは$\sum_{j=1}^{s} \tau_{i_j}$である．前と同様に，$G$内のタスク$T$のレベルは，$T$から$G$の終端タスクまでの最長の道の長さである．今回の定義は，全ての$i$に対して$\tau_i = 1$と仮定すれば，もとの定義と一致することに注意しよう．

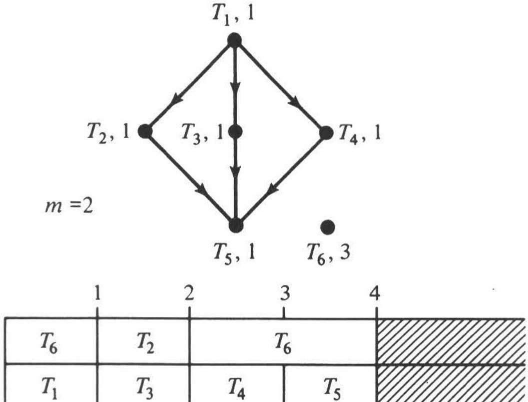



**図3.6-5** 横取り権あり 対 プロセッサ共有

**定理 3.7**

任意のグラフ $G$ および $m(\geq 1)$台のプロセッサに対し，最適の横取り権なし† および横取り権ありスケジュールの長さをそれぞれ $t_N$ および $t_P$ で表す．このとき，次式が成り立つ．

$$\frac{t_N}{t_P} \leq 2 - \frac{1}{m} \tag{3.6.2}$$

しかも，上式は可能な限りの最良の上界を示している††．

**証明**　$t_N$ の上界と $t_P$ の下界を比較しながら進めよう．

$\mathcal{S}$ を $G$ に対する任意のリスト・スケジュールとし，$\mathcal{S}$ における(全てのプロセッサの)空き時間の合計を $I$ で表す．補題3.3より，$G$ 内のタスクのある連鎖が存在して，あるプロセッサが $\mathcal{S}$ で空いているときは必ずこの連鎖内のタスクが他のプロセッサ上で実行中でなければならない．この連鎖の長さは，高々 $G$ の最長の道の長さ $L$ であるから次式を得る．

$$I \leq (m-1)L$$

つぎの等式

$$t_N = \frac{I + \sum_{T_i \in G} \tau_i}{m}$$

から(全てのスケジューリング規律に対して)，次式が得られる．

$$t_N \leq \frac{\sum_{T_i \in G} \tau_i + (m-1)L}{m} \tag{3.6.3}$$

さて，本章で議論したスケジューリング規律のどれに対しても，スケジュール $\mathcal{S}$ の長さは最長の道の長さよりも短くなり得ないし，スケジュール $\mathcal{S}$ は全てのプロセッサを四六時中働かせ続けるよりもさらに効率的ではあり得ない．ゆえに，

---

† **訳注**　ここではリスト・スケジューリングによる横取り権のないスケジュールに限定される．なお，3.1節の末尾を参照せよ．

†† しかし，実行できるタスクがあるのにプロセッサが空いていても良いという横取り権なしスケジュールに対しては式(3.6.2)は可能な限りの最良の上界ではない．図3.1-2と問題3.16，3.17参照．

(3.6.4)
$$t_P \geq \max\left\{L, \frac{\sum_{T_i \in G} \tau_i}{m}\right\}$$

式(3.6.3)と式(3.6.4)から，

$$\frac{t_N}{t_P} \leq \frac{\sum_{T_i \in G} \tau_i + (m-1)L}{m \cdot \max\left\{L, \frac{1}{m}\sum_{T_i \in G} \tau_i\right\}}$$

この比は，$(1/m)\sum_{T_i \in G} \tau_i = L$ であるときに最大になることが分るから，これを代入すると式(3.6.2)の結果が得られる．

これが実際に可能な限りの最良の上界であることは，例によって示すことができる．図3.6-6に，長さがそれぞれ $(2m-1+\varepsilon)$ および $(m+2\varepsilon)$ である最適な横取り権なしおよび横取り権ありのスケジュールに対するグラフが示されている．したがって，

$$\frac{t_N}{t_P} = \frac{2m-1+\varepsilon}{m+2\varepsilon}$$

となり，$\varepsilon$ の選び方次第でいくらでも式(3.6.2)の上界に近づけることができる．　■

定理3.4と同じ見地に立つと，横取り権ありスケジュールから横取り権を取除いたときの最大の影響が，やはり式(3.5.1)($m' = m$ として)で表現される一つの“変則”を作るものと見なすことができるのは興味深い．

特定の横取り権のあるスケジューリング・アルゴリズムを紹介する前に，実行時間が全て等しいタスクのグラフに対して3.3節で述べたスケジュール長および最小のプロセッサ数の下界(式(3.3.1)および式(3.3.2))を拡張しよう．基本的には，必要なことはレベルの一般化である．$T_i$ から終端タスクまでの最長の道の長さから $T_i$ の実行時間を差引いたものを $L'(T_i) = L(T_i) - \tau_i$ と表す．$y(0 \leq y \leq \tau_i)$ に対し，$T_i$ は $L'(T_i) + y$ 以上のレベルに$(\tau_i - y)$単位の処理量を持つという．$x$ 以上のレベルの $G$ 内の総処理量を $W_x$ で表し，次のように定義する．

$$\Omega = \{T_j | T_j \in G \text{ かつ } L'(T_j) \leq x \leq L'(T_j) + \tau_j\}$$

とし，

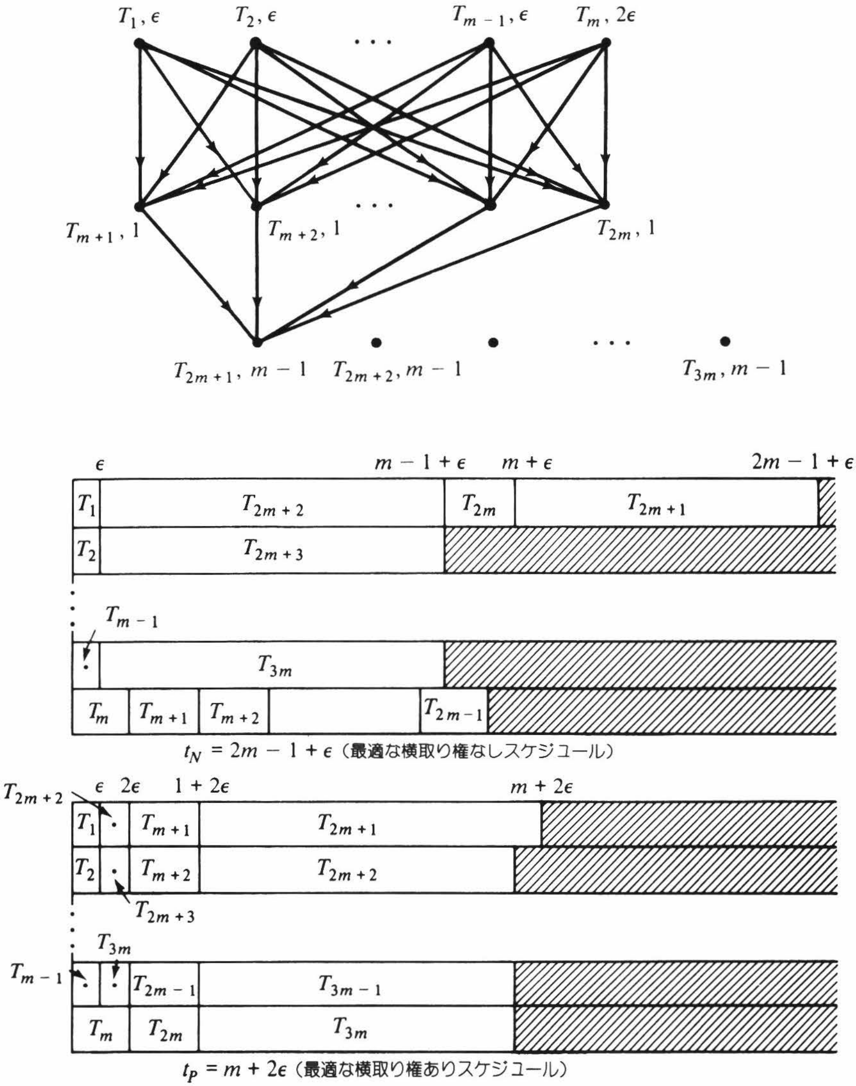



**図 3.6-6**　横取り権あり 対 横取り権なしスケジューリング

$$\Lambda = \{T_r | T_r \in G \text{ かつ } L'(T_r) > x\}$$

とする．このとき，次式が成り立つ．

(3.6.5) $$W_x = \sum_{T_r \in \Lambda} \tau_r + \sum_{T_j \in \Omega} [L(T_j) - x]$$

この一般化されたレベルの概念を用いて，式(3.3.1)と式(3.3.2)を確立するために用いたのと同じ議論が展開できる[11]. たとえば，$L$ を与えられたグラフの最長の道の長さとしたとき，$t\ (\geq L)$ 時間単位内でグラフを実行するために要する最小のプロセッサ数の下界は次式で与えられる.

(3.6.6) $$m_{\min} \geq \max_{0 \leq x \leq L} \left\{ \frac{W_x}{t - x} \right\}, \quad (t \geq L)$$

ここで一般化されたレベルを表す $x$ は，閉区間[0, $L$]の上の連続変数である. プロセッサ数を与えられた台数 $m$ に固定したときの計算時間の下界も同じ様にして一般化できる†. これらの新しい下界は，これまで論じてきたどのスケジューリング規律に対しても，また，タスクの実行時間が任意であるグラフに対しても有効である. これらの下界は，横取り権ありおよび一般のスケジューリング規律に対しては明らかに式(3.3.1)および式(3.3.2)よりも良い. タスクの実行時間が等しい場合には，この新しい下界は前の離散型版と同じ結果になることを確かめるのは難しいことではない.

さて，タスクの実行時間が任意であるという点を除いて3.2節または3.3節の条件が成り立つならば，最適な横取り権ありスケジューリングを生成する††アルゴリズムを紹介しよう. すなわち，任意のグラフに対するプロセッサ2台の場合および木構造のグラフに対する任意のプロセッサ台数の場合を考える. この横取り権ありのアルゴリズムは，平たくいうと，アルゴリズムAとBの極限，すなわち，$A$-スケジュールと $B$-スケジュールを生成するアルゴリズムとして開発されるであろう. アルゴリズムAから開始し，以下に述べるような極限をとる操作に基づいて最終的に得られるアルゴリズムは，木構造かつプロセッサ $m(\geq 2)$ 台の場合にも適用されることが分る.

実行時間がそれぞれ $\tau_1, \tau_2, \ldots, \tau_n$ であるタスクの任意のグラフ $G$ があると仮定する. $\tau_i (1 \leq i \leq n)$ は，**互いに通約的**(mutually commensurable)である，すなわち，ある正の実数 $w$ が存在して各タスクの実行時間は $w$ の整数倍であるという条件の下に任意であると仮定する. (今後，$w$ はそのような数の中の最大のものとしよう.) この通約性の仮定は，実際的な見地からは大して重

† 訳注 すなわち，$t_{\min} \geq \max_{0 \leq x \leq L} \left\{ x + \frac{W_x}{m} \right\}$.

†† 訳注 定理3.6によって，最適な横取り権ありスケジュールの生成は，同時に，最適な一般スケジュールの生成をも意味する.

要ではない．なぜならば，$\tau_i$ の任意の集合は，互いに通約的な集合でいくらでも密に近似できるからである．

$w$ を用いて，各タスク $T_i(\tau_i = n_i w)$ を $w$ 時間単位の実行時間を持つ $n_i$ 個の部分タスク $T_{i_1}, \ldots, T_{i_{n_i}}$ の連鎖で置換えることによって $G$ から得られるグラフを $G_w$ と定義する．たとえば，図 3.6-7(**a**)において，$w = 1$ と選ぶと図 3.6-7(**b**)のグラフ $G_1$ が得られる．$G_w$ に対する最適の横取り権なしスケジュール(たとえば，$m = 2$ のときはアルゴリズム A を使って得られる)は，横取りが時刻 $w, 2w, 3w, \ldots$ でしか許されていないシステムを仮定したときの $G$ に対する横取り権ありのスケジュールと考えることができる．図 3.6-7 には，各単位区間の終りでの割込みを許したために比率 7/6 の改善が可能になった $G$ の最適スケジュールも示してある．

次に，任意のプロセッサ数 $m(\geq 2)$を仮定して，(整数)$r \to \infty$ のときのグラフ $G_{w/r}$ に対するアルゴリズム A で生成される順序付けを考えよう．アルゴリズム A は決定するべき各時点では常に実行可能な最高レベルのタスクを最初

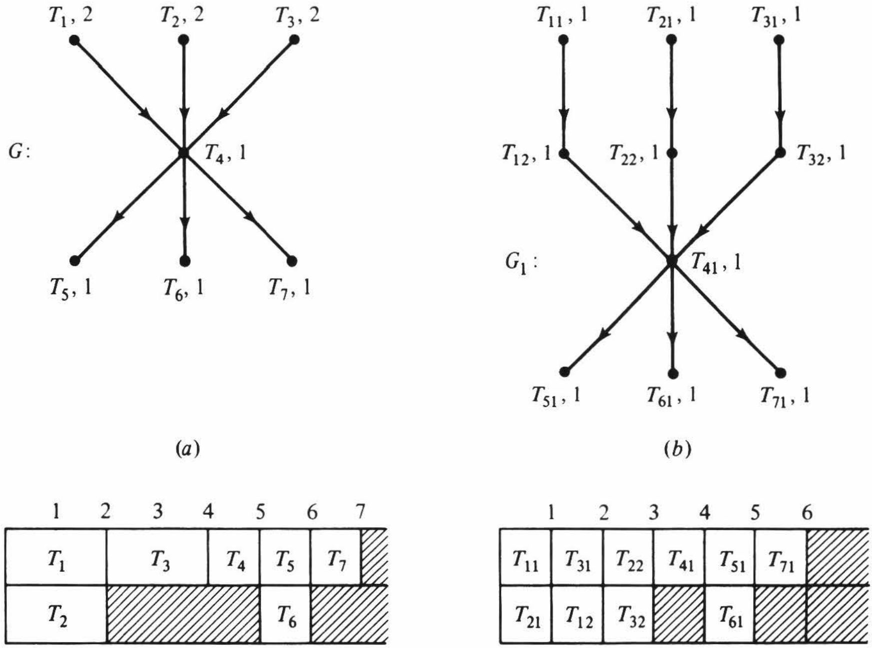



図 3.6-7　$G_w$ の例

に順序付けすることを思い出そう．$G$ の与えられたタスクは，そのスケジューリング優先度を $G_{w/r}$ においては各時間間隔 $w/r$ の上でしか保持できないから，$G_{w/r}$ における同レベルの部分タスクに対して付与されたラベルの順位は $w/r$ が小さくなるにつれてあまり意味を持たなくなってくる．したがって，最高レベル順という性質から，極限においてアルゴリズム A は，各時点においてグラフ $G$ の未実行タスクのうち，最高レベルにある全てに最高の優先度を与えるようなアルゴリズムに収束することは明らかである．$w/r \to 0$ であるから，最高レベルのタスクが二つ以上あるときは，これらのタスクが前述のプロセッサ共有の方式で $m$ 台のプロセッサを効果的に共有することも分る．特に極限においては，次の**一般**スケジューリング・アルゴリズムを得る．

**アルゴリズム C**

最高レベルのタスクからプロセッサを 1 台ずつ割当てる．最後の $a$ 台のプロセッサに対して $b$ 個$(a < b)$のタスクが(同じレベルにあるために)互角になっているならば，$a/b$ 台のプロセッサをこれらの $b$ 個のタスクに割当てる．下記の二つの事象のいずれかが生じたときにはいつも，上記の規則に従ってグラフ $G$ の未実行の部分にプロセッサを割当て直す．

**事象 1：** あるタスクが完了する．

**事象 2：** もし現在の割当てを継続するとすれば，より低いレベルのあるタスクを他のより高いレベルのタスクよりも速い速度で実行することになるという時点に到達する．

定理 3.5 で述べた手続きに従って得られる等価な横取り権ありスケジュールと共に，**図 3.6-8** にその一例を示す．この図から，$t = 1$ で事象 2 が発生していることが分る(これがそのような事象の唯一の発生である)．スケジューリングの目的にとって，グラフ内の最長の道を**臨界道**(critical path)と呼ぶことは自然であり，この意味でアルゴリズム C を臨界道アルゴリズムと呼ぶことができる(問題 3.8 も参照のこと)．

3.3 節に関連して，木 $G_{w/r}$ に適用されたアルゴリズム B も $r \to \infty$ のとき，($G$ に適用された)アルゴリズム C に収束することは容易に示される．以上により，アルゴリズム C に関する次の定理を非形式的に示したことになる．

**定理 3.8**

アルゴリズム C は，$m = 2$ の場合またはグラフが木である場合には最短ス

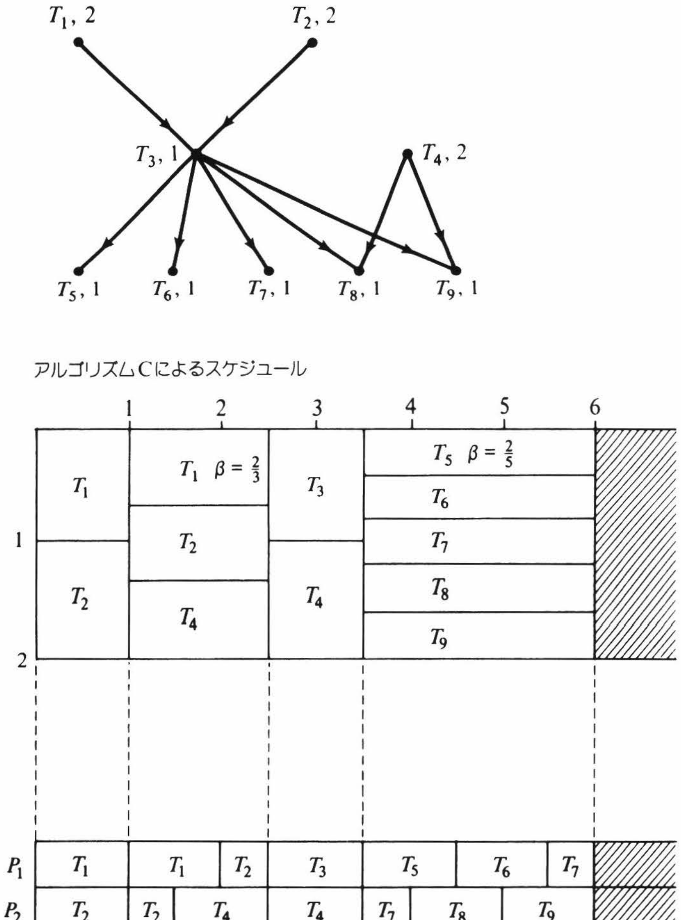



**図 3.6-8**　アルゴリズム C の例

ケジュールを生成する．

この定理の詳細な証明は実に長い [10, 12] ので，前述の極限をとる操作を与えた議論で止めておく．任意のグラフにおいて，プロセッサ $m(>2)$ 台に対してアルゴリズム C で与えられるスケジュールが必ずしも最適になるとは限らないことを示す例が図 3.6-9 に与えられている．アルゴリズム C に従え

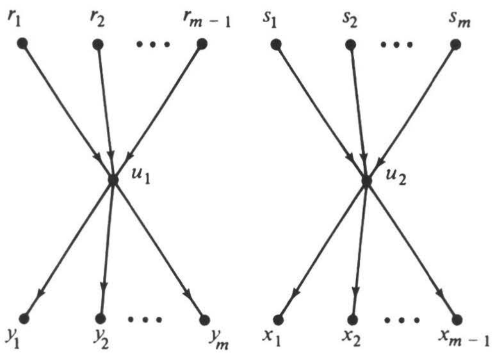



図3.6-9　アルゴリズムCの反例

ば，$s_1, \ldots, s_m$ および $r_1, \ldots, r_{m-1}$ は全て並行して実行されることになる．その後，$u_1$ と $u_2$ の同時実行があり，引続いて $x_1, \ldots, x_{m-1}$ および $y_1, \ldots, y_m$ の並列実行がある．これは $5-2/m$ のスケジュール長になるのに対し，最適スケジュールは長さ4によることは容易に分る．

まだ言及していない問題が一つあり，それを以下に示す．ある $k \geq 1$ に対し，$G_{w/(r+k)}$ の最適な横取り権なしスケジュールが，$G_{w/r}$ の最適な横取り権なしスケジュールよりも短いようなグラフ $G$ を，各 $r \geq 1$ について見つけることができるであろうかという問題である．プロセッサ2台の場合についてはその答は否であることが知られている．特に，$G_{w/2}$ の最適な横取り権なしスケジュールは，$G$ の最適な横取り権ありスケジュールと正確に同じ長さになることが知られている(この事実を理解するために図3.6-7と図3.6-8参照)．したがって，プロセッサが2台の場合に対しては，任意のグラフの最適な横取り権ありスケジュールを計算する代替手段があることになる．アルゴリズムAが $G_{w/2}$ に適用されるか，アルゴリズムCが $G$ に適用されるかである．プロセッサ $m$ $(>2)$ 台への拡張として，$G$ の最適な横取り権ありスケジュールは，$G_{w/m}$ の最適な横取り権なしスケジュールと長さが同じであると推測されている[11]．

## 3.7　役割の異なるプロセッサから成るシステム

役割がそれぞれ異なる多数のプロセッサによってタスク・システム(ジョブ)が処理されていくような重要な応用が沢山ある．中央処理装置(CPU)群と入

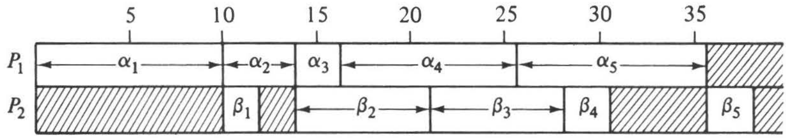
出力装置群から成るシステムはその典型的な例である．それぞれ異なるプロセッサを必要とし，ジョブ毎に処理順序が指定されている高々 $m$ 個のタスクから成るタスク・システムの集合を考える．これに対応する最短スケジュールを求める問題は，全てのタスクの実行時間が予め分っていることを前提にすると，産業への応用が多数あることから，通常，ジョブ・ショップ問題(job-shop problem)といわれている．この一般の問題を解決する数え上げ法でない手続きは，$m=2$ という自明ではない最も単純な場合に対してさえも，まだ知られていない．しかし，同じように重要な他の組合せ問題と同様に，発見的手法，分岐限定法(branch-and-bound)，および様々の計画法的な手法が効果的に適用されてきている．分り易い議論が文献[2]でなされている．

ジョブ・ショップ問題の一つの重要かつ特別な場合が生ずるのは，プロセッサには線形順序が付けられ，与えられたタスク・システムのタスクはこの線形順序によって定まる順に実行されなければならないということを仮定したときである．つまり，たとえば，入力タスクは最初に入力プロセッサ上で実行され，CPU タスクが次に実行され，最後に出力タスクが出力プロセッサ上で実行されることを仮定する必要がある場合があろう．ジョブ・ショップ問題のこの特殊化は，フロー・ショップ問題(flow-shop problem)として良く知られている．この問題に対して知られている最も一般的かつ効率的な解法 [13] は，$m=2$ に限定されている．この古典的な結果を示した後，この型の解法が適用し易く，しかも計算機への応用で意味を持つような二つの特殊な拡張を述べよう．

プロセッサ2台のフロー・ショップ問題に対し，$\{C_i|1\leq i\leq n\}$をそれぞれ二つのタスクから成る連鎖の集合とし，$C_i$ の第1および第2のタスクの実行時間を $A_i$ および $B_i$ で表す．$C_i$ の第1のタスクはプロセッサ $P_1$ 上で，第2のタスクはプロセッサ $P_2$ 上で実行されなければならない．プロセッサ $P_1$ および $P_2$ に割当てられた $i$ 番目のタスクの実行時間をそれぞれ $\alpha_i$ および $\beta_i$ と表したときのスケジュールの一例が図 3.7-1 に示されている．($\alpha$ および $\beta$ の系列は，それぞれ，$A_1, A_2, \ldots, A_n$ および $B_1, B_2, \ldots, B_n$ の単なる置換である)．まず，プロセッサ2台の場合について，吟味する必要があるスケジュールのクラスを著しく狭い範囲に限定できることを示そう．

**補題 3.4**

プロセッサ2台のあらゆるフロー・ショップ・スケジュールから次の性質を持つ高々同じ長さのスケジュールを構成することができる．




$(A_i, B_i)$: $(2\frac{1}{2}, 7)$ $(10, 1\frac{1}{2})$ $(9, 2)$ $(10, 2)$ $(4, 7)$

$(\alpha_i, \beta_i)$: $(A_2, B_2)$ $(A_5, B_5)$ $(A_1, B_1)$ $(A_3, B_3)$ $(A_4, B_4)$

**図 3.7-1** スケジュール例

1. どちらのプロセッサもそれによって実行可能なタスクがあるときは決して空きにはならない(空き時間を導入する必要は全くない).
2. 2台のプロセッサ上に現われるタスク・システムの系列は互いに同じである(すなわち，第1および第2のプロセッサ上の $i$ 番目のタスクは同じタスク・システムに属している).

**証明** ここでは，2を証明し，1の証明は読者に任せよう．

図3.7-2(**a**)に示すように，$C_i$(の第1タスク)が $C_j$(の第1タスク)に $P_1$ 上では先行しているが $P_2$ 上では後続しているスケジュール $\mathcal{S}$ があるものと仮定する．明らかに，先行制約を乱すことなく，$P_1$ 上のタスクを並べ替えて $\mathcal{S}$ を図3.7-2(**b**)のスケジュール $\mathcal{S}'$ に変換することができる．すなわち，$C_i$ と $C_j$ のタスクが隣り合うようにこれらのタスク間にある全タスクを左へ動かす．最後に，$\mathcal{S}'$ は図3.7-2(**c**)のスケジュール $\mathcal{S}''$ に変換できることは一目で分る．したがって，帰納的に，両プロセッサ上でタスク・システムの系列が同じであるスケジュールを構成することができる． ∎

第1の性質から，考える必要のあるスケジュールは，$P_1$ 上の割当てが図3.7-1のように左詰めに揃えられたものだけで良いことが分る．第2の性質からは，2-プロセッサ問題では，同一系列が $P_1$ と $P_2$ の両方に適用されるという了解の下に一つの系列について考えれば良い(実際，そうすることになる)．補題3.4の第1の性質は，任意のプロセッサ数に対して容易に一般化される．第2の性質は，プロセッサ $P_{m-1}$ と $P_m$ に対して対称的に適用することで少しばかり一般化できる．すなわち，$m$-プロセッサのフロー・ショップ問題に対し，$P_1$ と $P_2$ の上のタスクの系列は同じにできるし，さらに $P_{m-1}$ と $P_m$ 上の系列も同じにできる．3-プロセッサの場合は(2-プロセッサの場合と同様に)，全てのプロセッサ上で同じタスク・システムの系列を持つスケジュールだけを

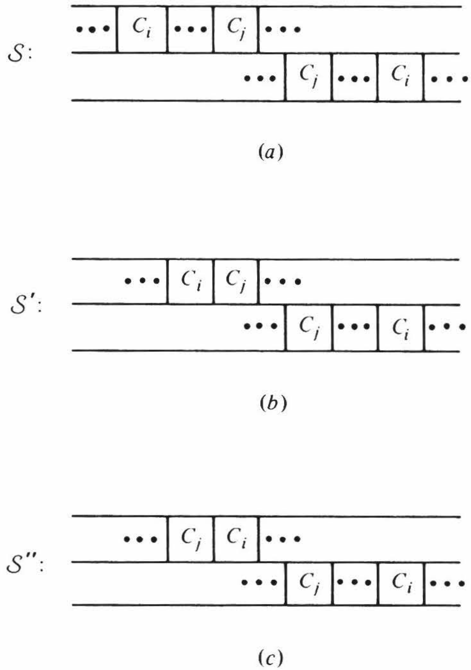



図 3.7-2　$P_1$ と $P_2$ 上に同じ系列を作り出す方法

考えれば良い．問題 3.15 は，この“順序保存”の性質が $m > 3$ のときの全プロセッサに拡張できないことを証明する $m = 4$ に対する例を示している．次に主要な結果を示そう．

**定理 3.9**

$\mathcal{S}$ を 2-プロセッサのフロー・ショップ問題に対する一つのスケジュールとし，$\mathcal{S}$ は補題 3.4 で述べられている性質を持つものと仮定する．全ての $i$ および $j$ に対して，$\min(A_i, B_j) < \min(A_j, B_i)$ であるときはいつも $C_i$ が $C_j$ に先行しているならば，$\mathcal{S}$ は最短である．

**証明**　$P_2$ 上の(内部の)空き時間が最小であることを示すと $\mathcal{S}$ が最短であることが証明されることになる．したがってまず最初に，この空き時間を $\alpha_i$ や $\beta_i$ を用いて表現することから始めよう．

$P_1$ 上で $i$ 番目のタスクが実行している間の $P_2$ の空き時間を $X_i$ と定義する．図 3.7-1 の例では，$X_1 = 10$, $X_2 = 2\frac{1}{2}$, $X_3 = 0$, $X_4 = 0$, $X_5 = 5\frac{1}{2}$ であ

る． $X_i > 0$ であるための必要十分条件は次式が成り立つことであることはすぐ分る．

$$\sum_{j=1}^{i-1} X_j + \sum_{j=1}^{i-1} \beta_j < \sum_{j=1}^{i} \alpha_j \tag{3.7.1}$$

このことから次式が得られる．

$$X_i = \max\left\{0,\ \sum_{j=1}^{i} \alpha_j - \sum_{j=1}^{i-1} \beta_j - \sum_{j=1}^{i-1} X_j\right\} \tag{3.7.2}$$

この式(3.7.2)を解くために，まず両辺に

$$R_{i-1} = \sum_{j=1}^{i-1} X_j \tag{3.7.3}$$

を加えると，次式を得る．

$$R_i = \max\{R_{i-1},\ Y_i\},\quad (i > 1) \tag{3.7.4}$$

ただし，$R_1 = Y_1 = \alpha_1$ であり，

$$Y_i = \sum_{j=1}^{i} \alpha_j - \sum_{j=1}^{i-1} \beta_j,\quad i > 1 \tag{3.7.5}$$

である．しかし，式(3.7.4)は，次の関数の漸化式であることが容易に理解できる．

$$R_i = \max\{Y_1,\ Y_2,\ \ldots,\ Y_i\} \tag{3.7.6}$$

このことから，$\mathcal{S}$ の長さは，$Y_i (1 \leq i \leq n)$ の最大(すなわち $R_n$)が最小化されるときに最短となる．$\min(A_i,\ B_j) < \min(A_j,\ B_i)$ のとき $C_i$ を $C_j$ に先行させるならば，$R_n$ が最小になることを証明しよう．簡単のため全ての $i, j$ に対し，$\min(A_i,\ B_j) \neq \min(A_j,\ B_i)$ であることを仮定して始めよう．

既に $P_1$ と $P_2$ 上に $(k-1)$ 個のタスク・システムを割当ててあり，スケジュール上で $k$ 番目と $(k+1)$ 番目の場所には $C_i$ と $C_j$ が考えられているものと仮定する．$C_i$ が $k$ 番目，$C_j$ が $(k+1)$ 番目の場所を占めると仮定するならば，式(3.7.5)から次式を得る．

$$Y_k = A_i - \beta_{k-1} + Y_{k-1};\qquad Y_{k+1} = A_j - B_i + Y_k \tag{3.7.7}$$

$C_j$ と $C_i$ の順序を入れ替えると，新しい系列に対して次式が得られる．

(3.7.8)　　$Y'_k = A_j - \beta_{k-1} + Y_{k-1}; \quad Y'_{k+1} = A_i - B_j + Y'_k$

式(3.7.7)と式(3.7.8)から，

$$\max(Y_k, Y_{k+1}) < \max(Y'_k, Y'_{k+1})$$

となるのは，

$$\min(-Y'_k, -Y'_{k+1}) < \min(-Y_k, -Y_{k+1})$$

となるとき，すなわち，

$$\min(-Y'_k, B_j - A_i - Y'_k) < \min(-Y_k, B_i - A_j - Y_k)$$

となるときである．全ての項に $A_j + Y_k$ を加えると，不等式は次のようになる．

$$\min(Y_k - Y'_k + A_j, Y_k - Y'_k - (A_i - A_j) + B_j) < \min(A_j, B_i)$$

式(3.7.7)と式(3.7.8)から得られる関係 $Y_k - Y'_k = A_i - A_j$ を代入すると，次式を得る．

(3.7.9)　　$\min(A_i, B_j) < \min(A_j, B_i)$

したがって，式(3.7.9)が成立するならば，$C_i$ を $C_j$ の前に置いたときに上記の二つの $Y_i$ の値の大きい方が($C_j$ を $C_i$ の前に置いたときに比べて)より小さくなる．このことから，任意のスケジュールにおいて条件(3.7.9)に違反しているような隣接したタスクの位置を入れ替えることにより，$\max\{Y_1, Y_2, \ldots, Y_n\}$は減少するのみであるということになる．もしも式(3.7.9)によって定義される関係がタスク・システム間の線形順序を構成することを論証すれば，それで証明は完結する．なぜなら，これが論証された場合，定理の記述を満たすスケジュールは常に存在し，しかもただ一つでなければならないからである．しかしながら，この関係は，推移的であることさえ証明されれば明らかに線形順序である．この証明は少し面倒であるが，場合分けで行えるから，詳細は読者に任せよう†．

---

† **訳注**　以下に証明を述べる．
$C_i$ と $C_j$ が式(3.7.9)を満足するとき，しかもそのときに限り，$C_i \prec C_j$ と書こう．

**命題**　関係 $\prec$ は推移的である．すなわち，$C_i \prec C_j$ かつ $C_j \prec C_k$ ならば，$C_i \prec C_k$ である．

**証明**　まず，二つのタスク・システム $C_m, C_n$ に対して，関係 $C_m \prec C_n$ を用いて行った$(A_m, B_m)$↗

$\min(A_i, B_j) = \min(A_j, B_i)$である $i$ と $j$ が存在するとき，このような $C_i$ と $C_j$ のどちらが先行しても $\max\{Y_1, Y_2, \ldots, Y_n\}$は影響を受けないことが容易に確かめられる．したがって，一般には，二つ以上の最短スケジュールが存在し得る． ∎

計算機への応用にとっては，時間的に重ならない入力，実行および出力の3段階が順々に実行されるジョブ処理の形態を取扱えるように上述の結果を拡張することは明らかに望ましい．この拡張は，一定の限度まで可能である．特に，$m=3$ の場合を考え，$A_i$, $B_i$ および $D_i$ はそれぞれ $P_1$, $P_2$ および $P_3$ 上の実行時間であると定義する．タスク・システムは，プロセッサ $P_1$, $P_2$ および $P_3$ 上でこの順に実行されなければならないものと仮定する．上述の定理の証明とよく似た手法を使うことにすると，$\min\{A_i\} \geq \max\{B_i\}$または min

---

↘ と$(A_n, B_n)$の関係の場合分けを**表 C.1** に示す．たとえば，$\min(A_m, B_n) = A_m$, $\min(A_n, B_m) = B_m$ ならば，関係 $A_m < B_m \leq A_j$ が成立することが表 C.1 中の見出しの下の第2行に記入されている．

**表 C.1** 関係 $C_m \prec C_n$ による$(A_m, B_m)$と$(A_n, B_n)$の関係の場合分け

| $\min(A_m, B_n)$ | $\min(A_n, B_m)$ | $\min(A_m, B_n) < \min(A_n, B_m)$から得られる$A_m, B_m, A_n, B_n$の間の関係 |
|---|---|---|
| $A_m$ | $A_n$ | $A_m < A_n \leq B_m$ |
| $A_m$ | $B_m$ | $A_m < B_m \leq A_n$ |
| $B_n$ | $A_n$ | $B_n < A_n \leq B_m$ |
| $B_n$ | $B_m$ | $B_n < B_m \leq A_n$ |

次に，$C_i \prec C_j$, $C_j \prec C_k$ であるとき，表 C.1 を用いて得られる$(A_i, B_i)$と$(A_j, B_j)$の関係の場合分けおよび$(A_j, B_j)$と$(A_k, B_k)$の関係の場合分けの組合せに対してさらに得られる$(A_i, B_i)$と$(A_k, B_k)$の関係を**表 C.2** に纏めて表示する．たとえば，場合$(A_i < B_i \leq A_j)$と場合$(B_k < B_j \leq A_k)$の組合せに対して，$A_i < B_i$ かつ $B_k < A_k$ が成立していることが同表中に示されている．しかし，場合$(B_j < A_j \leq B_i)$と場合$(A_j < A_k \leq B_j)$の組合せ等に対しては，$B_j < A_j$ かつ $A_j < B_j$ という矛盾が生じるから，このような組合せは存在しない．

**表 C.2** $C_i \prec C_j$, $C_j \prec C_k$ であるときの$(A_i, B_i)$と$(A_j, B_j)$の関係の場合分けおよび$(A_j, B_j)$と$(A_k, B_k)$の関係の場合分けの組合せに対する$(A_i, B_i)$と$(A_k, B_k)$の関係

| | $A_j < A_k \leq B_j$ | $A_j < B_j \leq A_k$ | $B_k < A_k \leq B_j$ | $B_k < B_j \leq A_k$ |
|---|---|---|---|---|
| $A_i < A_j \leq B_i$ | $A_i < A_k$, $B_i$ | $A_i < A_k$, $B_i$ | $A_i < B_i$<br>$B_k < A_k$ | $A_i < B_i$<br>$B_k < A_k$ |
| $A_i < B_i \leq A_j$ | $A_i < B_i < A_k$ | $A_i < B_i < A_k$ | $A_i < B_i$<br>$B_k < A_k$ | $A_i < B_i$<br>$B_k < A_k$ |
| $B_j < A_j \leq B_i$ | $A_j < B_j$かつ$B_j < A_j$(矛盾)であるからこの組合せは存在しない | | $B_k < A_k < B_i$ | $B_k < A_k$, $B_i$ |
| $B_j < B_i \leq A_j$ | | | $B_k < A_k < B_i$ | $B_k < A_k$, $B_i$ |

表 C.2 を視察すれば分るように，矛盾の生じない全ての組合せに対して，$\min(A_i, B_k) < \min(A_k, B_i)$，すなわち $C_i \prec C_k$ が成立することは明らかである． ∎

$\{D_i\} \geq \max\{B_i\}$を仮定するならば,

$$(3.7.10)\qquad \min(A_i + B_i,\, B_j + D_j) < \min(A_j + B_j,\, B_i + D_i)$$

が成り立つときはいつも，$C_i$ を $C_j$ の前に置くことにより最短スケジュールが得られることが証明できる．つまり，この特別な場合には，$A_i$ が $A_i + B_i$ で，$B_i$ が $B_i + D_i$ で置換えられていることを除いて，定理3.9に示されているのと同じスケジューリング・アルゴリズムを用いれば良いのである．さらに，$P_2$ 上の実行時間は，$P_1$ または $P_3$ 上の実行時間以下に完全に抑えられているという条件が必要であることに注意しよう．もしも，$A_i$ および $D_i$ が入力および出力の処理に対応し，$B_i$ が CPU の活動を表すとすると，入出力制約のシステムを含むいくつかの重要な応用では上記の条件が極めて良く満たされると思われる．実験的な研究[2]では，$B_i$ が $A_i$ にも $D_i$ にも抑えられていない場合でさえも，式(3.7.10)は優れた“**第1近似**(first cut)”のスケジューリングの指針であることが示されているということは指摘しておくだけの価値がある．

最後に，Jackson[14]は，各タスク・システムの1番目と3番目のタスクは全て同じプロセッサ上で実行され，2番目のタスクはタスク毎に異なったプロセッサ上で実行されるようなモデルを吟味している．$A_i$, $B_i$ および $D_i$ を前と同じように定義すると，やはり式(3.7.10)の規準は最短スケジュールを導き出すということが証明されている．

## 3.8　平均流れ時間を最小にするスケジューリング

スケジュール全体の長さの他に，興味のある主なスケジュールの尺度は，平均の**流れ時間**(flow time, またはシステム内時間)である．より正確にいえば，$n$ 個のタスクを持つタスク・システム $C$ のスケジュールに対して，タスク $T_i$ の完了時刻を $t_i$ で表したとき，平均流れ時間は次式で定義される．

$$(3.8.1)\qquad \bar{t} = \frac{1}{n}\sum_{i=1}^{n} t_i$$

この尺度が重要であることの主な理由は，スケジュール開始時刻0から完了時刻 $t_{\max} = \max\{t_i\}$ までの区間上の未完了タスク数の平均 $\bar{n}$ との関連にある．($\bar{t}$, $t_{\max}$ および $\bar{n}$ はスケジュールが異なれば値も変り得るが，本節では曖昧さが生じない限りそれらの記号にこの依存関係を一々明示しない)．考察中の応用問題に対して，未完了タスクの数は時間の関数としての離散過程になる．特に，その数は $t = 0$ の $n$ から $t = t_{\max}$ の 0 まで，高々 $n$ 回離散的に変動する．

さて，与えられたスケジュールに対し，完了の順番によってタスクを順序付けよう．ただし，二つ以上のタスクが同時に完了する場合は，どちらを前にしてもよい．この順序に従ったタスクの完了時刻を $t(1) \leq t(2) \leq \cdots \leq t(n)$ で表す．$t(0) = 0$ と定義すると，次式を得る．

$$\overline{n} = \frac{1}{t_{\max}} \sum_{i=0}^{n-1} (n-i)\,[t(i+1) - t(i)] \tag{3.8.2}$$

これらの定義を説明する例が**図 3.8-1** に与えられている．

式(3.8.2)を二つの総和の形に展開すると次のように書ける．

$$\overline{n} = \frac{1}{t_{\max}} \left\{ \sum_{i=1}^{n} (n-i+1)t(i) - \sum_{i=1}^{n} (n-i)t(i) \right\} = \frac{1}{t_{\max}} \sum_{i=1}^{n} t(i)$$

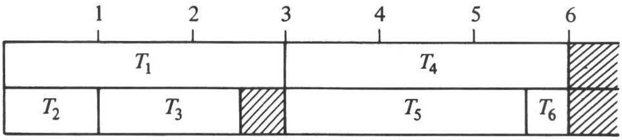
しかし，

$$\sum_{i=1}^{n} t(i) = \sum_{i=1}^{n} t_i = n\overline{t}$$

であるから，次式を得る．

$$\frac{\overline{n}}{n} = \frac{\overline{t}}{t_{\max}} \tag{3.8.3}$$

このように，平均未完了タスク数の最大タスク数に対する比は，平均流れ時間の最大流れ時間(スケジュール長)に対する比と同じになる．与えられたタスク




$t(i)$: $1,\ 2\frac{1}{2},\ 3,\ 5\frac{1}{2},\ 6,\ 6$

$t_i$: $3,\ 1,\ 2\frac{1}{2},\ 6,\ 5\frac{1}{2},\ 6$

$$\overline{t} = \frac{1 + 2\frac{1}{2} + 3 + 5\frac{1}{2} + 6 + 6}{6} = 4$$

$$\overline{n} = \frac{1}{6}\left\{6 \times 1 + 5 \times 1\frac{1}{2} + 4 \times \frac{1}{2} + 3 \times 2\frac{1}{2} + 2 \times \frac{1}{2}\right\} = 4$$

**図 3.8-1** $\bar{t}$ と $\bar{n}$ の例

の集合とスケジュール長に対し，式(3.8.3)は平均流れ時間が未完了タスク数の平均に直接比例することも示している．

この重要な関係は，スケジューリングのために全てのタスクが同時に提示されるという仮定を取除いた動的システムにも拡張される．とくに長期間走行において，完了したタスク数と到着したタスク数の(期間)平均が等しいという一般平衡性の仮定の下では，$\bar{t}$ と $\bar{n}$ との間の線形関係が成立する．この結果の持つ一般性は強調されるべきであろう．実効的には，プロセッサと順序づけのアルゴリズムを包含するシステムはブラック・ボックスと見なし得る．定常状態を仮定すると，ブラック・ボックスの内部構造やタスク自体の性質(および分っていると仮定されるそれらの特性)はこの結果の正しさに影響を与えないであろう．第4章では，この $\bar{t}$ と $\bar{n}$ の間の関係がある種の確率モデルの解析を簡単にするのに大いに役立つことになる．

単一プロセッサのスケジューリング問題では，空き時間を導入しても何の利益にもならない．したがって，式(3.8.3)の $t_{\max}$ は定数(タスクの実行時間の総和)であり，さらに平均流れ時間を最小(最大)化するどのような手続きもシステム内の平均のタスク数を最小(最大)化し，またその逆も成り立つ．以下では，単一プロセッサの特別な場合について，等価な問題である平均流れ時間最小化問題とシステム内平均タスク数最小化問題を解くことにしよう．最後に，前節までの多重プロセッサ・モデルに対して得られる同様の結果についての若干の注意で本節を締めくくろう．

プロセッサを1台に制限したとき，$\bar{t}$ と $t_{\max}$ を最小にするためには，タスクからの横取りを考える必要がないことを証明することは難しいことではない．空き時間を導入することも決して利益にならないから，**置換**(permutation)スケジュールのクラスのみを考察すればよいことになる．置換スケジュールは，実行されるべきタスクの置換によって完全に定義される．そこでは，与えられた置換によって決まる系列に従って左側から順に完了するようにタスクを一つずつ実行することが仮定されている．次のアルゴリズムDは，先行関係を与えている順序構造を1本の系列に織り込んで行くことによって，最小平均流れ時間を与える系列を効率的に決定する [2]．[一般化については問題3.21参照.]

$C$ を次式のような $r$ 本の独立な連鎖から成るタスク・システムとする．

(3.8.4)　$$C_i = T_{i1}T_{i2}\cdots T_{ip_i}(1 \le i \le r)$$

すなわち，各 $i$ に対して，$T_{i1} < T_{i2} < \cdots < T_{ip_i}$ である．$T_{ij}$ の実行時間を $\tau_{ij}$ で表し，$n = \sum_{i=1}^{r} p_i$ とする．

**アルゴリズム D**

**段 1.** 各タスクに対して，$\overline{n}_{ij} = (\sum_{h=1}^{j} \tau_{ih})/j$ を計算し，次に各連鎖に対して，$z_i(h_i) = \min[\overline{n}_{i1}, \ldots, \overline{n}_{ip_i}]$ を計算する．ただし，ここで $h_i$ は最小の値 $(\overline{n}_{ih_i})$ を指す指示子である．

**段 2.** $z_k(h_k) \leq z_i(h_i)\,(1 \leq i \leq r)$ であるような連鎖 $C_k$ を一本選び，部分連鎖 $T_{k1}T_{k2}\cdots T_{kh_k}$ を系列の先頭とする(または系列の後尾に付加する)．

**段 3.** $C_k$ から段2で割当てられたタスクを取除き，この新しい連鎖 $C_k$ に対する $z_k(h_k)$ を計算し直す．

**段 4.** 全てのタスクが系列に加えられるまで段2と段3を繰返す．

このアルゴリズムが平均流れ時間を最小にする系列を生成することを証明するのに次の補題が役に立つ．

**補題 3.5**

$\tau_1, \tau_2, \ldots, \tau_n$ は次の性質

$$\frac{1}{k}\sum_{i=1}^{k}\tau_i > \frac{1}{n}\sum_{i=1}^{n}\tau_i \qquad (1 \leq k < n) \tag{3.8.5}$$

を持った正整数であるとし，$w_1, w_2, \ldots, w_n$ は，$w_1 \leq w_2 \leq \ldots \leq w_n$ かつ $\sum_{i=1}^{n} w_i = 1$ を満たす非負の数とする．このとき，次式が成り立つ．

$$\frac{1}{n}\sum_{i=1}^{n}\tau_i \geq \sum_{i=1}^{n} w_i\tau_i \tag{3.8.6}$$

すなわち，式(3.8.6)の右辺の和は，$w_i = 1/n\,(1 \leq i \leq n)$ と選ぶことにより最大になる．

**証明** $w_1, w_2, \ldots, w_n$ を補題の条件を満たす重みの任意の集まりとし，$w_{k+1} = w_{k+2} = \cdots = w_n$ であるような最小の整数を $k \leq n-1$ と定義する．今，$k > 0$ であると仮定する．このとき，$w_1, \ldots, w_k$ をそれぞれ $\varepsilon$ だけ増やし，$w_{k+1}, \ldots, w_n$ をそれぞれ $k\varepsilon/(n-k)$ だけ減らすことができ，さらに，$0 \leq w_1 \leq w_2 \leq \cdots \leq w_n$ かつ $\sum_{i=1}^{n} w_i = 1$ という性質を保存するような最大の $\varepsilon > 0$ が存在することは明らかである†．これらの操作を行うことによって $\sum_{i=1}^{n} w_i\tau_i$ 内に生ずる変化は，

---

† **訳注** 最大の $\varepsilon$ を用いたこの増減操作によって $w_k = w_{k+1} = \cdots w_n$ となることに注意.

$$\Delta = \left[ \sum_{i=1}^{k} (w_i + \varepsilon)\tau_i + \sum_{i=k+1}^{n} \left( w_i - \frac{k\varepsilon}{n-k} \right) \tau_i \right] - \sum_{i=1}^{n} w_i \tau_i$$

すなわち,

$$(3.8.7) \quad \Delta = \varepsilon \left[ \sum_{i=1}^{k} \tau_i - \frac{k}{n-k} \sum_{i=k+1}^{n} \tau_i \right] = \varepsilon \left[ \sum_{i=1}^{n} \tau_i - \frac{n}{n-k} \sum_{i=k+1}^{n} \tau_i \right]$$

で与えられる．$\Delta \geq 0$ であることを証明するには,

$$\sum_{i=k+1}^{n} \tau_i = \sum_{i=1}^{n} \tau_i - \sum_{i=1}^{k} \tau_i = n \left[ \frac{1}{n} \sum_{i=1}^{n} \tau_i \right] - k \left[ \frac{1}{k} \sum_{i=1}^{k} \tau_i \right]$$

であることに気がつきさえすればよく，このことから式(3.8.5)を使うと次式を得る．

$$(3.8.8) \qquad \sum_{i=k+1}^{n} \tau_i \leq \frac{n-k}{n} \sum_{i=1}^{n} \tau_i$$

不等式 $\Delta \geq 0$ は，この式(3.8.8)から直接導かれる．この変換の($k$ 回の)繰返しは，$\sum_{i=1}^{n} w_i \tau_i$ が $w_i = w_{i+1} (1 \leq i \leq n-1)$ のときに最大になることを示しており，以上のことから $w_i = 1/n (1 \leq i \leq n)$ が得られる．■

**定理 3.10**

アルゴリズム D の対象として定義されたタスク・システムを考え，このアルゴリズムによって生成される系列を $\mathcal{P}_0$ で表す．このとき，$\mathcal{P}_0$ に対する平均流れ時間は最小である．

**証明**　タスク・システム $C$ に対する任意の最適な系列はどれも，平均流れ時間を増加させることなしに，アルゴリズム D によって生成される正当な系列に変換できるということをまず証明しよう．

$\mathcal{P} = T(1)\,T(2) \cdots T(n)$ は，アルゴリズム D によって最初に(段 **2** の **1** 回目の実行によって)生成される部分連鎖 $b$ がその先頭を占めていない最適な系列であるとし，$b = T^*_1 T^*_2 \cdots T^*_p$ とする．また，$\{\tau^*_i\}$をそれぞれの実行時間とする．$\mathcal{P}$ に関して，$b$ 内にないタスクが $b$ のタスク $T^*_{i-1}$ と $T^*_i$ の間で実行している時間間隔( 0 もあり得る)を $X_i (2 \leq i \leq p)$と定義する．また，$X_1$ を 0 から $T^*_1$ の開始時刻に至る間隔とし，$X_{p+1}$ を $T^*_p$ の終了時刻から $\mathcal{P}$ の完了時刻までに至る間隔とする．図 3.8-2 は，$p = 3$ の場合の例を示している．

さて，部分連鎖 $b$ を左に移動させて $b$ が変換後の系列 $\mathcal{P}'$ の先頭になるよ

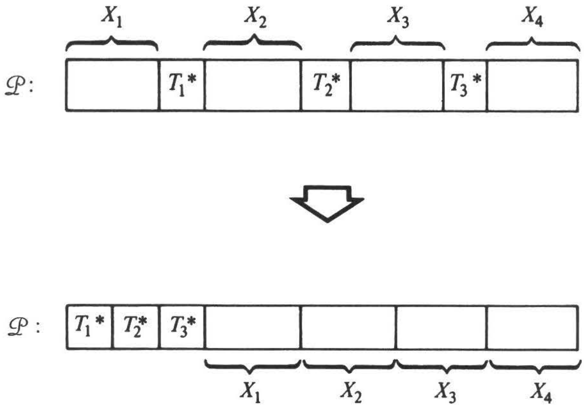



**図 3.8-2**　定理 3.10 の例($p=3$)

うにしよう．各間隔 $X_i$ はそれにつれて右に移動するが，それらの相対的な順序は変らない．この簡単な変換は図 **3.8-2** に図解されている．証明したいのは，$\mathcal{P}$ に対する平均流れ時間 $\overline{t}=(1/n)\sum_{i=1}^{n}t_i$ は，少なくとも $\mathcal{P}'$ に対する平均流れ時間 $\overline{t'}=(1/n)\sum_{i=1}^{n}t'_i$ と同じ大きさであることである．

最初に，$\overline{t'}$ を $\mathcal{P}$ の $t_i$ で表現する．このためには，$X_i$ 内のタスクの完了時刻は $\mathcal{P}$ 内の $t_i$ から $\mathcal{P}'$ 内の $t_i+\sum_{j=i}^{p}\tau^*{}_j$ に増えていることに注意しよう．図 **3.8-2** を参考にすると次式を得る．

$$
(3.8.9)\quad \sum_{i=1}^{n}t'_i=\sum_{i=1}^{p}(p-i+1)\tau^*{}_i+\sum_{t_j\in X_1}[t_j+\sum_{i=1}^{p}\tau^*{}_i]
$$

$$
+\sum_{t_j\in X_2}[t_j+\sum_{i=2}^{p}\tau^*{}_i]+\cdots+\sum_{t_j\in X_p}[t_j+\tau^*{}_p]\ +\sum_{t_j\in X_{p+1}}t_j
$$

$X_i$ 内で実行されるタスク数を $n_i$ と表し，$Y_j=\cup_{i=1}^{j}X_i$ とすると，式(**3.8.9**)は次のように書ける．

$$
(3.8.10)\quad n\overline{t'}=\sum_{i=1}^{p}(p-i+1)\tau^*{}_i+\sum_{t_j\in Y_{p+1}}t_j+\sum_{j=1}^{p}(n_j\sum_{i=j}^{p}\tau^*{}_i)
$$

前と同様に，$\mathcal{P}$ 内で実行される $j$ 番目のタスクの実行時間を $\tau(j)$ と表す．このとき，間隔 $X_i$ 内のタスクと $b$ 内のタスクとを区別して扱うと次のように書くことができる．

$$(3.8.11)\quad n\bar{t} = \sum_{t_j \in Y_{p+1}} t_j + [\tau^*_1 + \sum_{t_j \in Y_1} \tau(j)] + [\tau^*_1 + \tau^*_2 + \sum_{t_j \in Y_2} \tau(j)] + \cdots + [\tau^*_1 + \cdots + \tau^*_p + \sum_{t_j \in Y_p} \tau(j)]$$

$\Delta = n(\bar{t} - \bar{t'})$を計算すると次式を得る．

$$(3.8.12)\quad \Delta = \sum_{i=1}^{p} [(p-i+1) \sum_{t_j \in X_i} \tau(j)] - \sum_{i=1}^{p} (n_i \sum_{j=i}^{p} \tau^*_j)$$

今や，目標は $\Delta \geq 0$ を示すことである．要素(タスク)の終了時刻が間隔 $X_1, \ldots, X_p$ の和集合 $Y_p$ に含まれるような(各連鎖 $C_i$ の最大長の)部分連鎖の集合を $B$ とする．式(3.8.12)中の和 $\sum [(p-i+1) \sum \tau(j)]$ は $B$ 全体の上でとられており，項 $\tau(j)$ $(j \in Y_p)$ が全て使い尽されていることの認識から始めよう．言い換えると，式(3.8.12)は $B$ 内の部分連鎖の上の和に分割できる．式(3.8.12)において，$B$ 内の任意の部分連鎖に対応する部分[すなわち，任意の部分連鎖に限定した式(3.8.12)内の和であり，後述の $\Delta_i$]が非負であるならば，$\Delta$ は非負の項の和(非負)となる．そこで，$B$ 内の任意の部分連鎖に限定した式(3.8.12)が実際に非負であることを証明しよう．

$i$ 番目の連鎖 $C_i$ の次のような部分連鎖

$$(3.8.13)\quad T_{i1}T_{i2}\cdots T_{ik_1}\cdots T_{ik_2}\cdots T_{ik_p} \qquad (0 \leq k_1 \leq k_2 \leq \cdots \leq k_p)$$

が $B$ 内の部分連鎖の一つであると仮定する．ただし，便宜上 $k_0 \equiv 0$ とする．$T_{i(k_j+1)}\cdots T_{ik_{j+1}}$ は，$i$ 番目の連鎖の $X_{j+1}(0 \leq j \leq p-1)$ で実行される部分であるとし，もしも $k_{j+1} = k_j$ ならば，$X_{j+1}$ で実行する $i$ 番目の連鎖の部分はない($k_1 = k_2 = \cdots k_j = 0 (j \geq 1)$であることはその部分連鎖が $X_{j+1}$ 内で実行を開始することを意味する)．式(3.8.12)を部分連鎖(3.8.13)内の各タスクが寄与する項に特殊化するため，$n_i$ を $(k_i - k_{i-1})$ で置換える．そうすると，式(3.8.13)内のタスクのみに限定した和が次のように再構成される．

$$(3.8.14)\quad \Delta_i = \sum_{j=1}^{k_p} \tau_{ij} + \sum_{j=1}^{k_{p-1}} \tau_{ij} + \cdots + \sum_{j=1}^{k_1} \tau_{ij} - k_1\tau^*_1 - k_2\tau^*_2 - \cdots - k_p\tau^*_p$$

ところで，アルゴリズム D の記述から，

$$\frac{1}{k_h} \sum_{j=1}^{k_h} \tau_{ij} > \frac{1}{p} \sum_{i=1}^{p} \tau^*_i \qquad (1 \leq h \leq p)$$

すなわち，

$$\sum_{j=1}^{k_h} \tau_{ij} > \frac{k_h}{p} \sum_{i=1}^{p} \tau^*{}_i \qquad (1 \leq h \leq p) \tag{3.8.15}$$

式(3.8.15)の右辺を式(3.8.14)の各和に代入すると，$\Delta'_i \geq 0$ ならば $\Delta_i \geq 0$ であるということができる．ただし，

$$\Delta'_i = \left(\sum_{i=1}^{p} k_i\right) \frac{\sum_{i=1}^{p} \tau^*{}_i}{p} - \sum_{i=1}^{p} k_i \tau^*{}_i$$

明らかに，$w_i = k_i / \sum_{i=1}^{p} k_i$ と置いたとき，次式が成り立てば $\Delta'_i \geq 0$ である．

$$\frac{1}{p} \sum_{i=1}^{p} \tau^*{}_i \geq \sum_{i=1}^{p} w_i \tau^*{}_i \tag{3.8.16}$$

$w_1, \ldots, w_p$ は非減少であり，その総和は 1 である．さらに，補題 3.5 の式(3.8.5)の成立がアルゴリズム D の記述において仮定されている．したがって，式(3.8.16)の不等式，$\Delta'_i \geq 0$ および $\Delta_i \geq 0$ は全て成り立つ．

$\mathcal{P}$ 上で行った上述の変換を，アルゴリズム D の実行によって選ばれた 2 番目の部分連鎖に対してさらに $\mathcal{P}'$ 上で繰返すことができるということの考察がまだ残されている．帰納法を使うと，$\mathcal{P}_0$ を得るまでこの変換を繰返した後でも平均流れ時間が $\mathcal{P}$ のそれを越えないということが容易に確かめられる．■

アルゴリズム D は，$C$ が独立なタスクの集合から成っている特別な場合には全く簡単なものになる．特に，次の系を得る．

**系 3.1**

$\mathcal{P}_0$ は単一プロセッサに対する独立なタスクの(有限長の)系列であるとする．$\mathcal{P}_0$ に対応する実行時間の系列が非減少列であるならば，$\mathcal{P}_0$ の平均流れ時間は最小である．このような系列を作り出す機構は**最短処理時間順**(shortest-processing-time, SPT)規則と呼ばれている．

系 3.1 の別証は，次式のような重みつきの和(重み $(n-i+1)$ は減少列をなしている)が $\tau(1) \leq \tau(2) \leq \cdots \leq \tau(n)$ となるように系列を選ぶことにより最小化されるという良く知られた事実から得られる．

$$n\bar{t} = \sum_{i=1}^{n} (n-i+1) \tau(i)$$

一般には，タスクの連鎖に対して，どのタスクの終了時にも横取りが許されているとは仮定できないであろう．しかし，全タスクあるいは一部のタスクの終了時の横取り割込みが認められていないという比較的一般の場合には，与えられた条件によって終了直後に横取りが許容される部分連鎖に対してのみ $\overline{n}_{ij}$ を計算するようにアルゴリズムDの段1を改訂することができる．次の重要かつ特別な場合が得られる．

系 3.2

同じ連鎖のどの2タスク間でも横取りは許されていない(したがって，$\mathcal{P}$ は単なる連鎖の置換になる)と仮定する．$\mathcal{P}$ の平均流れ時間が最小化されるべきであるとすると，全ての $i, j$ に対して次式が成り立つならば，$\mathcal{P}$ 内で $C_i$ は $C_j$ に先行しなければならない．

$$\frac{1}{p_i}\sum_{k=1}^{p_i}\tau_{ik} < \frac{1}{p_j}\sum_{k=1}^{p_j}\tau_{jk}$$

SPTによる順序付けは，全タスクが独立であると仮定したときの多重プロセッサのモデルに容易に拡張できる．この特別な場合に対して，規則は次のようになる．プロセッサが空いたときはいつも，残っているタスクのうち実行時間が最小のものを割当てる．この規則は3.4節で議論した最大処理時間順(LPT)スケジューリング規則の正反対であることに特に注意しよう．全ての $m \geq 1$ に対して，LPTはたとえ最短ではないにせよ“まずまず”のスケジュールを生成するけれども，平均流れ時間を最小にするための規律としてはあり得る最悪のものとなる．後者のためにはSPTが最善であるけれども，特に実行時間に大きなばらつきがある場合に，スケジュール長を最短にすべく直観に頼ってLPTを用いたがる傾向があるのである．

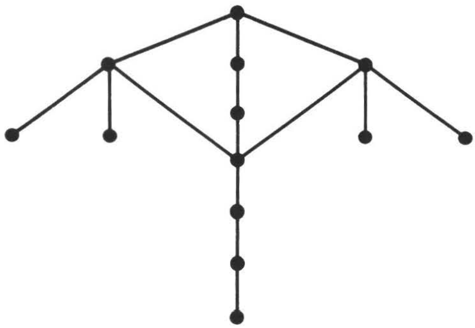
## 演 習 問 題

3.1 グラフ $G$ 内の $T$ のレベルを $L(T)$ と表す．$G$ は3.2節のアルゴリズムによってラベル付けされていると仮定する．$G$ 内の任意の二つのタスク $T$ と $T'$ に対して，$L(T) > L(T')$ ならば $\alpha(T) > \alpha(T')$ であることを証明せよ．

3.2 3.2節のプロセッサ2台の問題を考える．次の性質を持つ別のラベル付けアルゴリズムがあると仮定する．$T$ と $T'$ はラベル付きグラフ $G$ の任意の二つのタスクであるとする．このとき $|S(T)| > |S(T')|$ ならば $\alpha(T)$

$>\alpha(T')$が成り立つ．上記の性質を持つラベル付けアルゴリズムは最適でないことを示すグラフを見つけよ．

3.3 問題3.2に述べられた性質の代りに次の性質を持つ新しいラベル付けアルゴリズムを考えよう．$D(T)$は$T$の全ての後続タスクの集合を表すものとする．このとき，ラベル付きグラフ$G$の任意のタスクの対$T, T'$に対して$|D(T)|>|D(T')|$であるならば$\alpha(T)>\alpha(T')$が成り立つ．図P3.3のグラフは，そのようなラベル付けアルゴリズムの最適性に対する反例を与えていることを示せ．

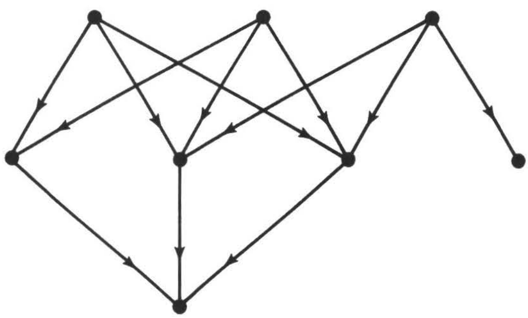


図P3.3

3.4 3.2節のプロセッサ2台の問題に対する次のようなもっと単純なアルゴリズムを考える．プロセッサが空いたとき，次に割当てられるタスクは必ず下記の条件を満たす未実行タスクの一つである．

1. 最高位のレベルにあり，かつ，
2. その後続タスクの集合は他の任意の未実行タスクの後続タスクの集合に真には含まれていない．

互角のものの順番を決めるために任意の規則を想定して良い．図P3.4の



図P3.4

グラフは，上記のアルゴリズムが必ずしも最短スケジュールを生成するとは限らないということを示している．このことを確かめよ．

**3.5** 与えられたグラフに対する2-プロセッサのラベル付けアルゴリズムの実行を考える．$S(T)$内のタスクのラベル付けが$S(T')$内のタスクのラベル付けより前に完了するならば$\alpha(T) < \alpha(T')$であることを示せ．この性質はアルゴリズムの次のような簡単化を可能にする．ある時点でラベル付け可能であるのに未だラベルの付いていないタスクの集合を$U$で表す．すなわち，全ての$T \in U$に対して$S(T)$内のあらゆるタスクはラベル付けされている．$U$に属するタスク間に，減少列$N(T)$ $(T \in U)$による(半)† 順序を導入し，これに従ってラベル付けを行うことができる．$U$の全てのタスクがラベル付けされてしまう前に，$U$のタスクのいかなる先行者も取り上げる必要はない．$U$内のすべてのタスクがラベル付けされた後で，新しい集合$U'$が定義される．この過程は全てのタスクがラベル付けされてしまうまで繰返される．

**3.6** 適当な代表としてのグラフのクラスを選び，このクラスに対しては，実行時間がタスク数$(n)$の2乗に比例するような2-プロセッサのラベル付けアルゴリズムを詳細に示せ．このアルゴリズムは，半順序が定義されるためには高々$n^2/4$までの枝しか持ち得ない(問題2.9より)という意味で，2-プロセッサのスケジューリング問題の複雑さの下界を提示していることに注意しよう．

**3.7** 任意の先行グラフ$G$と$m(\geq 1)$台のプロセッサを考え，$\mathcal{S}$を$G$に対する一つのスケジュールとする．$\mathcal{S}^R$を$\mathcal{S}$の**逆**(reverse)**スケジュール**と定義する．すなわち，$\mathcal{S}$の完了時点が$\mathcal{S}^R$の開始時点であり，$\mathcal{S}^R$内のタスクの系列は，$\mathcal{S}$を右から左に読むことによって作られる．$G$の全ての先行制約(枝の向き)を逆にして得られるグラフを$G^R$で表す．すなわち，$G$に対する半順序での$T_i < T_j$は，$G^R$に対する半順序では$T_j < T_i$となる．$\mathcal{S}$が$G$に対する最短スケジュールであるとき，しかもそのときのみ$\mathcal{S}^R$も$G^R$に対して最短であることを証明せよ．

**3.8** 任意のタスク・システムの各タスクにそのレベルをラベルとして付けるアルゴリズムを設計せよ．(3.6節で与えた一般化されたレベルの定義を使

---

† **訳注**　$T, T' \in U$に対し，$S(T) = S(T')$ならば$N(T) = N(T')$であり，$T$と$T'$の間には順序関係はない．したがって，この場合，$U$は(全)順序集合とはならない．

用せよ.)

このアルゴリズムは，プロセッサの台数が無制限のときのタスク·システムのスケジューリングに関係のあるいわゆる**臨界径路解析**(critical route analysis) [15] と密接な関連がある．あるタスクの実行時間を与えられた量だけ増やしたときにそのタスク・システムの最短実行時間もそれと同じ量だけ増えるならば，そのタスクは臨界的であるということにしよう．プロセッサ数に制限がなければ最長の道は全て臨界径路である．これらの道上のあらゆるタスクは臨界的であり，これらの道の長さが実現可能な最短スケジュールの長さを決定する．上で求められたラベル付けアルゴリズムを，与えられたグラフ内の全ての臨界径路を見つけるアルゴリズムに拡張せよ.

**3.9** 次のようなグラフ $G$ および対応するタスクのリスト $L$ を見つけることが問題である．すなわち，$G$, $L$ からあるタスク $T$(およびこのタスクを項として含む全ての先行関係)を取除いて得られる新しいグラフを $G'$，対応するリストを $L'$ としよう．もとの $G$ および $L$ に対するリスト・スケジュールよりも $G'$ および $L'$ に対するリスト・スケジュールの方が長いような例を見出せ．(この変則と，実行時間を減らすことによって生ずる変則との関係に注意せよ.)

**3.10** この問題で，式(3.5.1)が可能な最良の上界であることの証明が完結する．(図 3.5-6 は，優先順位リストの変更は$(2-1/m)$倍までのスケジュール長の変化を生じさせ得ることを示している.)

**a.** 実行時間を減らすと$(2-1/m)$倍にいくらでも近い値にスケジュール長を変化させ得ることを示すために図 3.6-6 を使用せよ．変更後の値として，図 3.6-6 で与えられている$\{\tau'_i\}$を用い，もとの値としては $\tau_i = 2\varepsilon\ (1 \leq i < m)$ と $\tau_i = \tau'_i (m \leq i \leq 3m)$を用いよ.

**b.** $\mathcal{T} = \{T_1, T_2, \ldots, T_{m(m-1)+2}\}$, $< = \{(T_1, T_2), (T_1, T_3), \ldots, (T_1, T_{m(m-1)+1})\}$かつ,

$$\tau_i = \begin{cases} \varepsilon & (i=1) \\ 1 & (2 \leq i \leq m(m-1)+1) \\ m & (i = m(m-1)+2) \end{cases}$$

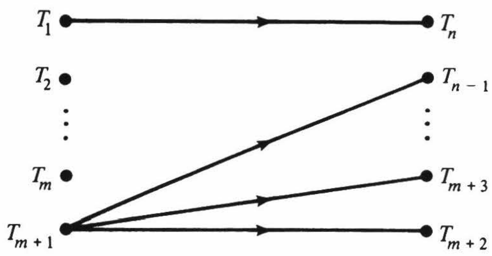
であるようなタスクシステム $C = (\mathcal{T}, <)$を考えよ．リスト $L = (T_1, T_2, \ldots, T_{m(m-1)+2})$を用いることにより，もしも $<$ 内の全ての先

行関係が取除かれたならばリストスケジュールの長さは$(2-1/m)$倍にいくらでも近い値に増加することを示せ.

**c.** 図P3.10に示したグラフにおいて, $n=mm'-m'+m+2$ かつ $m'>m>1$ とする. また, 実行時間は次のように与えられると仮定する.

$$\tau_i=\begin{cases}\varepsilon & (1\leq i\leq m+1)\\ 1 & (m+2\leq i\leq mm'-m'+m+1)\\ m' & (i=mm'-m'+m+2)\end{cases}$$

プロセッサ数が $m$ のときと $m'$ のときのスケジュール長の比は, 式(3.5.1)にいくらでも近づけることができるということを証明せよ.




図P3.10

**3.11** (納品時による順序づけ[2]). 単一プロセッサ上でスケジュールされるべき, 実行時間が$\{\tau_i\}_{i=1}^{n}$である $n$ 個の独立なタスクの集合を考える. (もし可能ならば)タスク $T_i$ が丁度終了**するべき**時刻であると解釈される $T_i$ の**納品時**(due date)を規定する正数を $d_i(1\leq i\leq n)$としよう. 任意に与えられたスケジュールに対して $T_i$ の完了時刻を $t_i$ で表すとき, $(t_i-d_i)$を $T_i$ の**遅延**(lateness)といい, $\max\{t_i-d_i, 0\}$をその**遅刻**(tardiness)という. 納品時の非減少順に従ってタスクを順序づけることにより, 最大のタスク遅延および最大のタスク遅刻は最小化されることを証明せよ. すなわち, $i$ 番目にスケジュールされたタスクの納品時を $d[i]$ で表すと, $d[1]\leq d[2]\leq\cdots\leq d[n]$ であることが遅延と遅刻の今述べた性質の成立を保証する. (定理3.9の証明で描いたような "交換" の論法を用いよ.) さらに, SPTによる順序づけが遅延の平均を最小にすることを確かめよ.

**3.12** 単一プロセッサと問題3.11で与えられた定義を考える. スケジュール $\mathcal{S}$ 内の任意の時点 $t$ において, $T_i$ の**猶予時間**(slack time)を$(d_i-\tau_i-t)$

で定義する．直観的には，任意に与えられた決断の瞬間において，それぞれの納品時に間に合わせるためには猶予時間が最小の未実行タスクを次にスケジュールすることが最も差迫っているように思われる．$t$ は猶予時間の相対的な関係には影響しないから，最小の猶予時間という基準を全体のスケジュールに適用することは，次の性質を持った順序づけを意味している．

$$d[1]-\tau[1] \leq d[2]-\tau[2] \leq \cdots \leq d[n]-\tau[n]$$

このような順序づけは，最小のタスク遅延および最小のタスク遅刻を最大にしてしまうという(いくらか期待外れの)結果になることを証明せよ．(問題 3.11 の証明と平行して行えば証明し易い．)

**3.13** 納品時と実行時間が下記のような四つのタスクの例を考える．

$$d_i\colon 1, 2, 4, 6$$
$$\tau_i\colon 2, 4, 3, 1$$

平均遅刻が最小になる単一プロセッサのスケジュールを示し，それがSPT，納品時順，猶予時間順のいずれの順序づけにも対応しないことを確かめよ．

**3.14** 実行時間を基準にするだけでなく，それとは独立なある優先度も基準にしてタスクを差別するために，次式で与えられる重みつき流れ時間の平均が最小になるように $n$ 個のタスクをスケジュールしたい．

$$\overline{t}=\frac{1}{n}\sum_{i=1}^{n} a_i t_i$$

ここで $t_i$ は完了時刻を表し，重み $a_i$ は相対的な重要度を示すように規定されている．このとき，

$$\frac{\tau[1]}{a[1]} \leq \frac{\tau[2]}{a[2]} \leq \cdots \leq \frac{\tau[n]}{a[n]}$$

であるようなスケジュールはこの重みつき流れ時間の平均を最小にすることを示せ．また，重みつきの遅延の平均も最小になることを確かめよ．

**3.15** 4 個のタスクからなる連鎖 2 本を次のように定義する．

| | $\tau_{i1}$ | $\tau_{i2}$ | $\tau_{i3}$ | $\tau_{i4}$ |
|---|---|---|---|---|
| $C_1$ | 4 | 1 | 1 | 4 |
| $C_2$ | 1 | 4 | 4 | 1 |

3.7節で述べたような4–プロセッサのフロー・ショップ・システム——すなわち，$T_{ij}$ はプロセッサ $j$ で処理されなければならないだけでなく，$T_{i(j-1)}$ が完了するまでは実行を開始できない——を仮定する．最適のフロー・ショップ・スケジュールの集合にはいつも順序保存型のスケジュール，すなわち，プロセッサ上の連鎖の系列は全て等しいというスケジュール(補題3.4参照)が含まれているという言明に対する反例を与えるために，上記で与えた2本の連鎖を用いよ．

3.16　式(3.6.2)で与えられている横取り権なしに対する横取り権ありのスケジュールの改善度の上界は，横取り権のないスケジュールにおいてタスクが実行可能であるときでもプロセッサを遊ばせておいてもよいとしたときには，最良の上界にはならない．(明らかに，任意の時点で全プロセッサが空いているという場合は考えなくてよい.)　このように制約がより少ない環境の下では，図3.6–6の横取り権なしスケジュールよりもかなり良い横取り権のないスケジュールを見つけることができることに注意しよう．このようなより一般的な場合には，任意のタスク・システムに対して次式が成立することを証明したい．

(P 1)
$$\frac{t_N}{t_P} \leq \frac{4}{3}$$

ただし，$t_N$ および $t_P$ は，それぞれ，最適の横取り権なしおよび横取り権ありのスケジュールの長さを表し，また，$m=2$ であるとする．まず，$n$–タスク・システムに対する $t_N$ とそれが必要とする全実行時間 $t_E = \sum_{i=1}^{N}\tau_i$ との大きさの関係に着目することから議論を始めよう．明らかに，$t_E \geq \frac{3}{2}t_N$ ならば，$t_P \geq t_E/2$ であるから(P1)の結果を得る．$t_E < \frac{3}{2}t_N$ のときも(P1)が成り立つことの証明が残されている．$I_N$ を全遊休時間とすると $t_N = \frac{1}{2}(t_E + I_N)$ であるから，$t_E < \frac{3}{2}t_N$ ならば $I_N > t_N/2$ であることが分る．

さらに，次のように議論を進めよう．

**a.** 与えられたグラフ $G$ の横取り権なしスケジュールによって，$G$ の連鎖 $C$ に与えられた並行性(concurrency)を，一方のプロセッサが $C$ のタスク上で動作中でありしかも他方のプロセッサも動作中である期間の合計時間として定義しよう．任意の連鎖 $C$ に与え得る並行性の最大は，$G$ のどのような横取り権なしスケジュールに対しても $t_N/2$ より大きくないことを示せ．

**b.** 補題3.3を使って，a. の一つの結果として次式が成り立つことを証明せよ．

$$I_P \geq I_N - \frac{t_N}{2}$$

ただし，$I_P$ は $G$ の最適な横取り権ありスケジュールの空き時間である．この不等式から(簡単な代数演算によって)(P1)が得られることを証明することがまだ残っている．(図3.6-1は，この上界が可能な最良のものであることを示している．)

**3.17** 問題3.16における一般的な接近法と補題3.3を用い，任意のプロセッサ数に対して次式が成り立つことを証明せよ．

$$\frac{t_N}{t_P} \leq \frac{2m}{m+1}$$

**3.18** 3.7節にあるJohnsonの2–プロセッサ・フロー・ショップ問題を一般化しよう．二種類あるプロセッサはそれぞれ2台以上あると仮定する．詳しく言えば，第1段の実行がなされるべき相等しい $A$ 型プロセッサが $m_1$ 台，第2段の実行がなされるべき相等しい $B$ 型プロセッサが $m_2$ 台あると仮定する．各タスク・システムは，実行が2段階で行われることに対応して丁度二つのタスクから成る1本の連鎖になっている．タスク・システム $C_i$ を，二つのタスクすなわち $A$ 型タスクおよび $B$ 型タスクの実行時間の対 $(A_i, B_i)$ で表す．

Shen と Chen[17]は，次の簡単なスケジューリング・アルゴリズムはやや能率が良いことを示した．$A_i + B_i \geq A_j + B_j$ かつ $A_i \leq A_j$ であるときしかもそのときに限り $C_i < C_j$ になるように，タスク・システムの集合 $\{C_i\}$ に先行関係(半順序) $<$ を導入する．(互角なものに対する先行関係の与え方にはどんな規則を仮定してもよい．) このとき，集合 $\{C_i\}$ は，$<$ を満たす任意の順序に従ってスケジュールされる．通例に従って，プロセッ

サはそれが実行するべき型のタスクが存在する限り，稼動し続けるものとする．集合{(1, 2)，(1, 2)，(2, 1)，(2, 1)，(2, 1)}は，$m_1 = m_2 = 2$であるとき上記で与えられたアルゴリズムが最適であるという主張に対する反例であることを示せ．(この例では，＜は$\{C_i\}$を**線型に**順序付けることに注意しよう．)

関係＜は，スケジュールされるべき集合$\{C_i\}$を線型に順序付けていると仮定する．$t_L$は上記の規則で達成される全実行時間であるとし，$t_{\min}$は達成可能な最短実行時間を表す．3.4節の方法を使って，もし$B$型プロセッサ上に中間の遊休期間がないならば次式が成り立つことを証明せよ．

$$\frac{t_L}{t_{\min}} \leq \frac{3}{2} - \frac{1}{2m_2}$$

また，最後の(それでスケジュールが完了する)$B$型タスクがそれに対応する$A$型タスクの終了直後に開始するならば，次式が成り立つことも証明せよ．

$$\frac{t_L}{t_{\min}} \leq \frac{3}{2} - \frac{1}{2m_1}$$

(後者の場合は$A$型プロセッサが混雑していることを想定しているが，前者の場合は$B$型プロセッサ上の混雑を想定している．)

**3.19**　実行時間が短くなったのにスケジュール長が長くなり得るといった変則を生じさせないような方法で，横取り機能をリスト・スケジュールに含ませる簡単な技法を考案せよ[11]．横取りは動的にも静的にも含ませ得ることに注意せよ．後者の場合，たとえば，全てのタスクは予め一連の小さな同じ大きさの副タスクに分割されていてもよかったのである．そのときには，これらの副タスクは横取り権なしで実行される．この種の静的横取りでは上記の変則を除去するのに十分ではないことが証明される．

**3.20**　与えられた独立なタスクの集合に対し，$t_{\mathrm{SPT}}$および$t_{\mathrm{LPT}}$はそれぞれ$m$プロセッサ上でのSPT, LPTスケジュールの長さとする．このとき次式が成り立つことおよびそれらが可能な最良の上・下界であることを証明せよ．(定理3.4を使うとよい．)

$$1 \leq \frac{t_{\mathrm{SPT}}}{t_{\mathrm{LPT}}} \leq 2 - \frac{1}{m}$$

3.21 (3.3節で定義されているような)木の集合を考える．部分連鎖ではなく部分木の節点の実行時間の正規化された総和を考えるという自然な方法により，アルゴリズムDを拡張せよ．この新しいアルゴリズムを詳細に記述した後で，このアルゴリズムが木の集合に対して生成する系列は平均流れ時間を最小にすることを証明せよ[18]．

## 参考文献


1. CLARK, W., *The Gantt Chart* (3rd ed.). Sir Isaac Pitman & Sons Ltd., London (1952).
2. CONWAY, R. W., W. L. MAXWELL, and L. W. MILLER, *Theory of Scheduling*. Addison-Wesley, Reading, Mass. (1967).
3. COFFMAN, E. G., JR., and R. L. GRAHAM, "Optimal scheduling for two-processor systems." *Acta Informatica 1*, 3 (1972), 200–213.
4. HU, T. C., "Parallel sequencing and assembly line problems." *Opns. Res. 9*, 6 (Nov. 1961), 841–848.
5. KRONE, M., "Heuristic programming applied to scheduling models," in *Proc. 5th Annual Conf. Inform. Sci. Sys.* Dept. of Electrical Engineering, Princeton Univ., Princeton, N. J. (1971), 193–196. See also, KRONE M., "*Heuristic programming applied to scheduling problems.*" Ph. D. thesis, Dept. of Electrical Engineering, Princeton Univ., Princeton, N. J. (Sept. 1970).
6. GRAHAM, R. L., "Bounds on multiprocessing timing anomalies." *SIAM J. Appl. Math. 17*, 2, (Mar. 1969), 416–429.
7. GRAHAM, R. L., "Bounds for certain multiprocessing anomalies." *Bell Sys. Tech. J.* (Nov. 1966), 1563–1581.
8. MANACHER, G. K., "Production and stabilization of real-time task schedules." *J. ACM 14*, 3 (July 1967), 439–465.
9. MCNAUGHTON, R., "Scheduling with deadlines and loss functions." *Manag. Sci. 12*, 1 (Oct. 1959). See also, ROTHKOPF, M. H., "Scheduling independent tasks on parallel processors." *Manag. Sci. 12*, 5 (Jan. 1966).
10. MUNTZ, R. R., and E. G. COFFMAN, JR., "Preemptive scheduling of real-time tasks on multiprocessor systems." *J. ACM 17*, 2 (Apr. 1970), 324–338.
11. MUNTZ, R. R., "Scheduling of computations on multiprocessor systems: The preemptive assignment discipline." Ph. D. thesis, Dept. of Electrical Engineering, Princeton Univ., Princeton, N. J. (Mar. 1969).
12. MUNTZ, R. R., and E. G. COFFMAN, JR., "Optimal preemptive scheduling on two-processor systems." *IEEE Trans. Comp. C-18*, 11 (Nov. 1969), 1014–1020.

13. JOHNSON, S. M., "Optimal two- and three-stage production schedules with set-up times included." *Nav. Res. Log. Quart. 1*, 1 (Mar. 1954).

14. JACKSON, J. R., "An extension of Johnson's results on job-lot scheduling." *Nav. Res. Log. Quart. 3*, 3 (Sept. 1956).

15. MODER, J. J., and C. R. PHILLIPS, *Project Management with CPM and PERT.* Van Nostrand Reinhold, New York (1964).

16. LIU, C. L., "Optimal scheduling on multiprocessor computing systems." *Proc. 13th Switching Automata Theory Symp. IEEE.* (1972), 155–160.

17. SHEN, V. Y., and Y. E. CHEN, "A scheduling strategy for the flow-stop problem in a system with two classes of processors," in *Proc. 6th Annual Conf. Inform. Sys. Sci.*, Dept. of Electrical Engineering, Princeton Univ., Princeton, N.J. (Mar. 1972).

18. SIDNEY, J. B., "One machine sequencing with precedence relations and deferral costs," Working Paper #125, Faculty of Commerce and Business Administration, University of British Columbia (1972) (to appear, *Journal of Operations Research Society of America*).

19. ULLMAN, J. D., "Polynomial completeness of the equal execution time scheduling problem." Princeton Univ., Dept. of Elec. Engrg., Computer Science Report TR-115 (Dec. 1972).

20. KARP, R. M., "Reducibility among combinatorial problems." Dept. Computer Science, Univ. Calif. at Berkeley, TR-3 (Apr. 1972).

# 計算機における順序付け問題の確率モデル

# 4

## 4.1 はじめに

### 4.1.1 基本定義

計算機操作の確率モデルによる解析は，オペレーティング・システムの予備的な研究にとって役に立つ技法となっている．数学的な接近法として，確率モデルは，一般に，決定性モデルよりもさらに現実的である．なぜならば，そうしたモデルによって，計算機利用者から発せられる不規則で予想し難い要求が表現可能になるからである．これらの要求は，オペレーティング・システムが補助記憶並びに主記憶装置，諸プロセッサ，入出力諸装置，および諸システム・ルーチン用に形成しかつ制御する多数の待ち行列から構成されている構造体のどこかに(待ち行列への到着者として)組み込まれるから，この意味で，確率モデルを用いて動的スケジューリングの諸技法の特性を研究しようとするならば，モデルを待ち行列システムの形で定式化しなければならないということになる．

本章では，プロセッサ待ち行列のスケジューリングに注意を集中し，それ以外の待ち行列活動は一般的に設定される実行諸条件に含まれているという巨視的な見方を取る．(第5章では入出力待ち行列に関する諸問題が取扱われる．)本章の目的は，それらの実行諸条件に関し知られていると仮定される情報を利用してシステム性能を向上させるタイムシェアリング・アルゴリズムおよびある種のバッチ処理アルゴリズムを研究することにある．このような研究は，モデルを数学的に扱いやすくしておくために理想化したり単純化したりする仮定を設けなければならないにもかかわらず，新しい順序付けアルゴリズムの特性を洞察する助けとなり得る．これらの数学的モデルはシミュレーション研究の妥当性の確認を助けるためにも定式化されるであろうし，またある場合には実在

システムの性能の直接の尺度あるいは予測を得るために——これまたモデル化の過程で単純化の必要性が見出されるにもかかわらず——使用することもできよう．本節の残りの部分は，プロセッサ・スケジューリングのモデル化に的を絞って適当に特殊化された数学的な待ち行列システムを特徴付けるのに当てることにする．待ち行列理論のここでの取扱い方は入門的であり，よく引用される諸文献[1—8]に見られるものである．

図 4.1-1 に示されるように，待ち行列システムの主な構成要素には，**扱い者**(サーバ, server)，滞留する客を収容する待合室あるいは記憶領域でできている**待ち行列**(queue)，およびシステムの利用者または客の集団である**入力源**(source)の三者がある．計算機への適用では，扱い者は，通常，1 個あるいはもっと多くのプロセッサから成っている．待ち合わせ設備としては，入出力装置，補助記憶，あるいは主記憶領域がある．客としては，たとえば磁気テープ上に一括して集められたジョブ，あるいは計算機システムにオンライン結合している利用者から発せられる様々な型の**サービス要求**(service request)がある．本章を通じて，**サーバ/プロセッサ**，**サービス/実行**(execution)/**操作**(operation)，また**到着者**(到着, arrival)/**ジョブ**という用語が，同じ語の不快な反復使用を最小にするためにも，適宜置き換えて使用される．利用者と(サービス)要求という用語は，タイムシェアリングの議論において適切であるような場合に用いられよう．

待ち行列システムの確率モデルを数学的に記述するには，引き続く到着の間の時間(到着時間間隔という)とジョブの走行時間(すなわち，サービス時間)を記述する確率分布関数をそれぞれ指定し，さらに**待ち行列の規律**(queueing discipline)を指定することが必要である．待ち行列の規律または**サービス規律**(service discipline)という言葉で，簡潔に，ジョブが実行される順序(sequence)と一つのジョブがサービスのために選び出される度にそれにどれだけの実行時間を割当てるかを決定する(オペレーティング・システム中に組み込まれている)規則あるいは手続きを表す．

到着時間間隔とサービス時間は統計的に互いに独立であるということが一貫

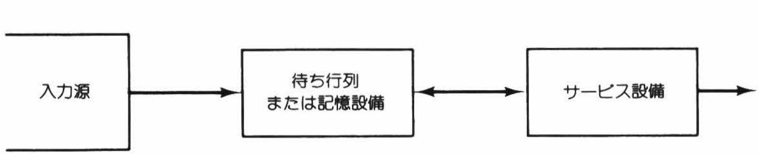



図 4.1-1　待ち行列システム

して仮定される．また，4.3 節を除いて，引き続く到着時間間隔と引き続く到着者へのサービス時間のどちらについても，各々が独立であることを仮定する．到着時間間隔とサービス時間に対応する確率変数をそれぞれ $T$ と $S$ で表す．到着時間間隔とサービス時間の分布関数に対して，それぞれ記号 $A(x) = \Pr[T \leq x]$, $B(x) = \Pr[S \leq x]$を使用する．確率変数 $T$ と $S$ は離散的であってもよいし，また連続的であってもよい．後者の場合には，対応する密度関数(存在するとき)を示すのに小文字を使用する．たとえば，$f(x) = dF(x)/dx$ は分布関数 $F(x)$に対応する密度関数を表している．

本章では，密度関数の存在をほとんど常に仮定するにもかかわらず，一般に，分布関数 $F(x)$を持つ非負の確率変数の $i$ 次モーメントを

$$\mathrm{E}(X^i) = \int_0^\infty x^i f(x)\,dx$$

と表現するよりもむしろスチルチェス積分

$$\mathrm{E}(X^i) = \int_0^\infty x^i dF(x) \qquad (i = 1, 2, \cdots)$$

を用いて表現するのを普通とする．より一層一般の分布への適用可能性を別にしても，そうすることによって表記上の経済性と確率変数の記述を分布関数に限定できるという利点が生じるからである．

### 4.1.2 到着過程

解析する全てのモデルを通じて $A(x)$は次式で与えられる(負の)**指数分布**であることが仮定される．

(4.1.1) $$A(x) = 1 - e^{-\lambda x} \quad (x \geq 0)$$

対応する密度関数は

(4.1.2) $$a(x) = \lambda e^{-\lambda x} \quad (x \geq 0)$$

である．最初の二つのモーメントを計算すると次の結果を得る．

(4.1.3) $$\mathrm{E}(T) = \frac{1}{\lambda}$$

(4.1.4) $$\mathrm{E}(T^2) = \frac{2}{\lambda^2}$$

これから分散は次式のようになることが分る.

$$\text{(4.1.5)} \qquad \mathrm{Var}(T) = \frac{1}{\lambda^2}$$

後になると，指数分布のパラメータがシステム状態の関数となる到着過程を考察することになる．しかしながら，今は，$\lambda$ がシステム状態に独立であると仮定しよう．独立かつ指数分布にしたがう到着時間間隔の仮定は待ち行列理論の恐らく唯一の最も特徴的な仮定である．そこで，時間をかけてその性質をもっと詳細に調べることにする.

まず，指数分布のいわゆる**無記憶性**(memoryless property)または**マルコフ性**(Markov property)について調べよう．それはこの仮定の核心的な重要性を納得させる性質である．ある事象が起こるのを待ち始める時点で，待ち時間の分布が式(4.1.1)におけるようにパラメータ $\lambda$ を持つ指数分布であることが分っていると仮定しよう．今，$t$ 時間単位の間待った後で，その事象が起こるまでさらに待たなければならない残りの時間を支配する分布関数 $R_t(x)$ が知りたいとする．値が全待ち時間を表す確率変数を $T$ と書くと，次式が成り立つ.

$$R_t(x) = \Pr[T \leq x + t | T > t] = \frac{\Pr[t < T \leq t + x]}{\Pr[T > t]}$$

式(4.1.1)を用いると

$$R_t(x) \equiv \frac{\int_t^{t+x} \lambda e^{-\lambda y} dy}{\int_t^{\infty} \lambda e^{-\lambda y} dy} = \frac{e^{-\lambda t}[1 - e^{-\lambda x}]}{e^{-\lambda t}}$$

これから

$$\text{(4.1.6)} \qquad R_t(x) = 1 - e^{-\lambda x}$$

こうして，$R_t(x)$ は $t$ に独立であって元の待ち時間分布に等しいという結果が得られる†．明らかに，システムは時間の経過とともに‘歳’を取らないのである.

無記憶性は，到着過程の言葉で述べられる次の問題においても示される．到

† 指数分布はこの無記憶性を持つ唯一の連続分布であることもそれほどの困難もなく示し得る.

着は，到着時間間隔が独立であって共通の分布関数(4.1.1)を持つ過程に従って起こるとしよう．任意ではあるが固定した$t > 0$から次の到着までの時間の分布はどのようなものになるであろうか．今度も，この分布は$t$に無関係であって式(4.1.1)で与えられる．この結果は，後に，事象間の時間が一般分布を持つという仮定の下で対応する結果を計算するときに確かめられる．要約すると，指数到着時間間隔の仮定は，新しい到着があるまでその後待たなければならない時間がそれまでに既に費やした待ち時間の長さに統計的に独立であるということを意味している．

指数分布の離散的同類は**幾何分布**である．例として，表が出る確率は$p$，裏が出る確率は$1-p$である一連の独立な硬貨投げ(ベルヌーイ試行)を考えよう．硬貨のどの一投から開始しても裏が出るまでに連続して表が出る回数$N$の分布は幾何分布

$$\Pr[N=n] = (1-p)p^n \quad (n=0, 1, \cdots)$$

であって，前回いつ裏が出たかには無関係である．これは，任意に与えられた一投げで裏が出る確率は$1-p$であり，その前に行われた一投げの結果には無関係であるという性質から来ている．こうして，到着時間間隔が独立で幾何分布する離散時間事象(到着)における無記憶性が得られる．このような到着はいわゆる**ベルヌーイ到着過程**(Bernoulli arrival process)を構成する．

連続時間で定義した上述の到着過程は，$\lambda$を簡便に**到着率**(arrival rate)と呼んでいるよく知られた**ポアソン過程**(Poisson process)である．さて，ここでポアソン過程のもう一つの定義を与えてそれが到着時間間隔に対する指数分布を導くことを示そう．$o(\Delta t)$は$\Delta t$より小さい位数を持つ任意の量を表すものとする．どんな微小時間$\Delta t$に対しても区間$(t, t+\Delta t)$内に到着のない確率が$1-\lambda\Delta t + o(\Delta t)$となり到着が一つある確率が$\lambda\Delta t + o(\Delta t)$となるような定数$\lambda$が存在するとしよう．ただし，区間$(t, t+\Delta t)$内の事象は$t$に統計的に独立であり，また重なり合わない他のどんな区間内の事象にも統計的に独立であるとする．上記の性質を満足する過程がポアソン過程であり，対応する到着時間間隔の分布は次のように計算することができる．任意の時刻$t_0$に対して，次の到着の時刻$x$が$t_0+t$を過ぎる確率を$A_c(t)$で表そう．すなわち，$A_c(t)$は$(t_0, t_0+t)$内に到着がない確率である．$z = x - t_0$と置くと，次式が成り立つ．

$$A_c(t+\Delta t) = \Pr[z > t+\Delta t]$$

あるいは

(4.1.7)
$$A_c(t+\Delta t) = \Pr[z > t]\Pr[\Delta t \text{ 内に到着がない}] = A_c(t)[1 - \lambda\Delta t + o(\Delta t)]$$

ここで互いに重複しない区間における独立事象の仮定を使用した．式(4.1.7)を変形し，$\Delta t$ で割り，$o(\Delta t)$ と同位の項を無視すると

$$\lim_{\Delta t\to 0}\frac{A_c(t+\Delta t) - A_c(t)}{\Delta t} = -\lambda A_c(t)$$

したがって，

(4.1.8)
$$A'_c(t) = -\lambda A_c(t)$$

境界条件 $A_c(0)=1$ の下では式(4.1.8)の解は次式の通りである．

$$A_c(t) = e^{-\lambda t} \quad (t \geq 0)$$

これは明らかに $t_0$ に無関係であり

$$A(t) = 1 - A_c(t) = 1 - e^{-\lambda t} \quad (t \geq 0)$$

である．以上は，到着時間間隔の指数分布がポアソン過程を定義する前述の性質から導かれることを示している．

最後に，時間間隔 $t$ 内に $i$ 個のポアソン到着がある確率を指定する**離散的確率塊関数**†(pmf, probability mass function)$f_i(t)$ $(i=0, 1, 2, \ldots)$を決定する方法を簡単に示す．この確率は区間がどの時点から始まるかに無関係であるという事実を直接に利用する．$t = m\Delta t$ と置き，互いに重複しない区間の統計的独立性を使い，さらに長さ $\Delta t$ の $m$ 個の区間のうち $i$ 個に一つずつ到着があり残りの $m-i$ 個の区間のそれぞれには到着がない確率を計算する．下に示すように2項確率の極限を取ることにより $f_i(t)$ が見出される．

$$f_i(t) = \lim_{\Delta t\to 0}\binom{m}{i}[\lambda\Delta t + o(\Delta t)]^i[1 - \lambda\Delta t + o(\Delta t)]^{m-i} \qquad \left(m = \frac{t}{\Delta t}\right)$$

これは次のように整理することができる．

$$f_i(t) = \frac{(\lambda t)^i}{i!}\lim_{m\to\infty}\frac{m(m-1)\cdots(m-i+1)}{m^i}\lim_{m\to\infty}\left[1 - \frac{\lambda t}{m}\right]^m$$

† **訳注**　連続分布の確率密度（probability density）に対応する離散分布の用語が確率塊（probability mass）である．

これから次式を得る.

$$(4.1.9)\qquad f_i(t) = \frac{(\lambda t)^i}{i!}e^{-\lambda t}$$

式(4.1.9)は**ポアソン分布**(Poisson distribution)を定義する. 到着時間間隔の分布は $A(t) = 1 - f_0(t)$ で与えられることに注意しよう. 式(4.1.9)から

$$(4.1.10)\qquad f_i(\Delta t) = \begin{cases} 1 - \lambda\Delta t + o(\Delta t) & (i = 0) \\ \lambda\Delta t + o(\Delta t) & (i = 1) \\ o(\Delta t) & (i > 1) \end{cases}$$

が容易に確かめられ, このようにポアソン過程に対して与えた定義間の無矛盾性が示される. 式(4.1.9)の平均値は $\lambda t$ になることが容易に計算される. 別の言い方をすれば, 時間間隔 $t$ 内の平均到着数は(平均)到着率 $\lambda$ に時間間隔の長さを掛けたものである. ポアソン分布の2次モーメントは $\lambda t(1 + \lambda t)$ で与えられ, このことから分散は平均と同じく $\lambda t$ であることに気付く. これまでに示した性質に基づく理由によりポアソン到着はしばしば**ランダム**(random)到着と呼ばれる.

並列ポアソン到着過程の二つの重要な性質が後の役に立つ. 図4.1-2(**a**)ではパラメータ $\lambda_i (i = 1, 2, \ldots, k)$ を持つ $k$ 個の独立なポアソン'流れ(stream)'の**合流**(confluence)または**和**(sum)が示されている. 結果の過程もまた到着率 $\lambda' = \lambda_1 + \lambda_2 + \cdots + \lambda_k$ を持つポアソン過程となる. 和の過程において微小な時間間隔 $\Delta t$ 内に到着がある確率は $\lambda'\Delta t + o(\Delta t)$ でありまた到着がない確率は $1 - \lambda'\Delta t + o(\Delta t)$ であることに気付くことが容易にこの結果に到る緒口となる.

定常確率分布 $\{\pi_i\}_{i=1}^k$ に従うポアソン過程の**分解**(decomposition)が図4.1-2(**b**)に示されている. **源流**(source)過程から到着が生じる度毎にそれは**支流**(branch)過程の一つに割当てられる. 到着が $i$ 番目の支流に割当てられる確率は $\pi_i (1 \leq i \leq k)$ である. 支流過程がそれぞれ到着率 $\lambda\pi_i (1 \leq i \leq k)$ を持ち, しかも互に独立なポアソン過程となることの証明は, 簡単であるから, 演習問題として残しておく.

### 4.1.3 サービス機構

本章で解析されるモデルの多くは任意のサービス時間分布が仮定されるけれども, 二つの重要なモデルで指数分布が再び特別の役割を演じる. 特に, 後者のこの二つのモデルにとって指数分布の持つ無記憶性は必要不可欠である. そ

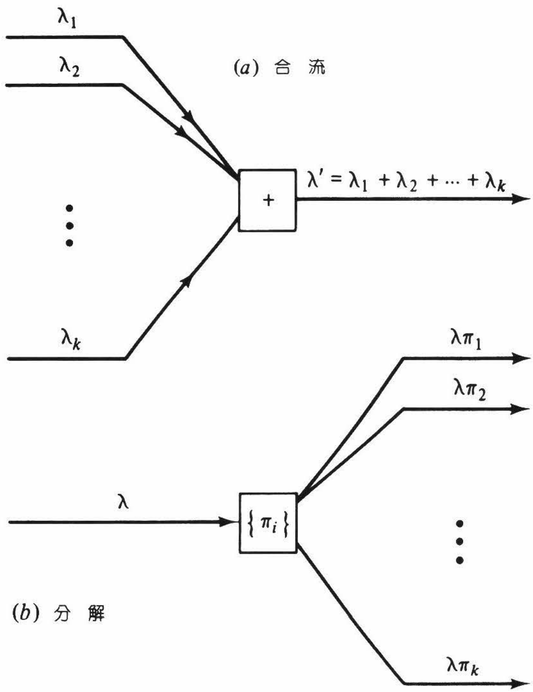



**図4.1-2**　ポアソン過程の組合せ

れというのも，システムを観察するために選んだ時点で，任意に与えられたジョブが，それまでにどれだけの量のサービスを受けているかにかかわりなく，その後なお必要とするサービスの量(すなわち，時間)の分布関数は元の指数分布に等しいと言明することをこの性質が可能ならしめるからである．

定義しなければならない待ち行列システムの第三の要素は**サービス規律**(service discipline)である．初めに，**横取り割込み可**(preemptive)**の規律**と**横取り割込み不可**(nonpreemptive)**の規律**を区別しよう．横取り割込み不可の規律の下では，ジョブは，一度実行を開始したならば，完了するに到るまで走行し続けなければならない．これに対し，横取り割込み可の規律の下では，任意の時点で，実行中のジョブを中断させてプロセッサ上から排除することができると仮定される．割込まれたジョブは待ち行列または記憶装置内に戻され，後刻，もう一度サービスの順番が回ってきたときに，再度プロセッサを割当てられる．横取り割込みの回数については全く制限を設けない．

サービス規律が使用する情報の量は広範囲に変化し得るが，勿論のこと，利用可能とされる情報が何であるかに依存する．計算機への応用において使用される情報は，実行時間，すでに受取った実行時間，到着時刻，記憶および入出力に関する諸要件†，ジョブの重要度あるいは緊急度を反映する優先権等についてあらかじめ得られる知識という形を取る．本書では，下記に関する情報を利用するサービス規律に範囲を限定する．

1. **到着時刻** たとえば，普通の**先着順**(first-in-first-out, FIFO)規則は単純に到着の順番に従ってジョブを実行する．
2. **ジョブ実行時間** 例としては，次に実行するジョブは現に待っているジョブの中で実行時間の最も短いものとするという**最短処理時間順**(shortest-processing-time, SPT)規則が挙げられる．
3. **単純優先権** 本章で解析されるこの基本的優先権モデルは，システム内にあるジョブがそれぞれその到着時点において(または，その時点以前に)優先権クラスを割当てられるということを仮定する．優先権クラスの集合は可算的であると仮定される．このような優先権は，記憶領域要求条件等を基にしてジョブ間に格差を付けるのに使用することができる．しかしながら，優先権割当ての機構にはあまり深入りしないであろう．なぜならば，第一に，特定の施設の目的と制約条件にあまりにも強く依存するからである．

上に述べた規律は，他のいくつかの規律と共に，後節でもっと完全に定義される．本節は，計算機システムに対する待ち行列モデルの通常の解析で追求される性能の尺度について簡単に論じて締めくくることにする．

### 4.1.4 性能の尺度

通常，計算されるべき第一のそして最も基本的な性能(または輻輳)の尺度は．時間の関数として求められるシステム(すなわち，サーバと待ち行列)内到着者(ジョブ)数の分布である．しかしながら，興味の対象となる多くのシステムに対して，この計算の複雑さは大変なものであり，しかも結果は解釈を与えるのが困難な形を取りがちである．そこで，普通は，定常確率分布を算出し，それを用いて長時間走行(long-run)または定常状態における振舞を求めるといった所に落着く．後者の結果は，一般にはるかに単純であり，その上多くの場合，システムの過渡的な振舞はほとんど興味を持たれない程十分に短時間のもので

† **訳注** たとえば，記憶領域必要量．

あるという事実によって正当化される.

利用者の観点から見ておそらく最も重要であると思われる性能の尺度は，利用者がシステム内で費やさなければならない時間の分布である．この分布は，実行時間に関する何らかの分っている尺度を基にしてジョブを差別するようなシステムでは，必要とするサービスの量によって条件付けられるのが普通である．(今後，条件付待ち時間といえば，必要とされる実行時間によって明確に条件付けられた待ち時間を指すものとする.)行列内における待ち時間の分布についても，この分布が上記(システムにおける待ち時間)の分布とは本質的に異なっているようなシステムについては興味がある．後の数節で考察されるモデルでは，関心は主に条件付待ち時間分布の平均値を見出すことに注がれる.

よく研究の対象となる第三の輻輳の尺度は，**全稼働期間**(busy period)の長さの分布である．単一サーバ・システムの場合，全稼働期間は，システムが空の状態にあるときのジョブの到着時点からシステムが再度空になる瞬間までの期間をいう†．本章で解析されることになるモデルに関する限り，全稼働期間の分布はいずれも同じである．本章では，この全稼働期間の平衡状態における分布の(ラプラス)変換に関する関数方程式が導出されることになる．なお，1次および2次のモーメントの計算は演習問題で扱われる.

次の節では，応用待ち行列理論における諸結果の中で最も基本となるものだけを取上げて述べることにしよう．その目的は，第一に，この分野での経験に欠けている読者に対し,計算機におけるスケジューリング問題(3.4節～3.8節)の確率モデルへの基本的な応用を研究するのに十分な予備知識を提供することにある．また，これらの結果は，他の資源割当て問題について第5章および第6章で用いられる技法の多くを理解するための背景として有用な知識を提供しもするのである.

特に待ち時間の計算において，待ち行列の理論は，独立な確率変数の和を扱うことがしばしばある．このことは，後続のいくつかの節で使用されるラプラス変換が極めて便利であるのはなぜなのかということの主な原因の一つになっている．また，一方，離散的確率分布を扱う場合，特に確率過程を記述する差分方程式を解く際に母関数($z$-変換)が多用される．これらの重要性を考慮して，補遺Aで，確率の理論にとり差当り必要な母関数とラプラス変換の定義と基本的な諸性質について述べておく.

---

† **訳注**　複数サーバの場合は，全てのサーバが busy になっている期間を指すか，1個でも busy になっている期間を指すか，あまりはっきりしていない．なお，本書では，この複数サーバの場合は扱われていない.

## 4.2 待ち行列に関する基本的結果

### 4.2.1 M/M/1 型待ち行列システム

ポアソン到着と次式に従って指数分布するサービス時間を伴う単一プロセッサ待ち行列を考える.

$$B(x) = 1 - e^{-\mu x} \quad (x \geq 0) \tag{4.2.1}$$

1次と2次のモーメントはそれぞれ $\mathrm{E}(S) = 1/\mu$ と $\mathrm{E}(S^2) = 2/\mu^2$ である. 主目標は統計的平衡状態におけるシステム内ジョブ数を記述する確率分布を求めることである. 得られる結果は, ジョブの実行順序を決めるのに実行時間に関するどんな予知情報も使用しない任意のサービス規律に対して有効である. そのような規律の例には FIFO, LIFO(後着順, last-in-first-out) ランダム選択(random sequencing), および, 優先権の割当てがジョブの実行時間に独立に行われる優先権規律が含まれる.

待ち行列の用語では上で定義したシステムは **M/M/1 型待ち行列システム**(到着時間間隔およびサービス時間に関する指数分布の仮定ゆえにポアソン型待ち行列とも呼ばれる)のクラスの一員である. この記法では, 1番目の文字は**到着時間間隔の分布**を, 2番目の文字は**サービス時間の分布**を, 3番目の文字は**サーバの数**をそれぞれ表す. M は(マルコフ性を持つ)**指数分布**, G は**一般分布**, D は**一定到着時間間隔**または**一定サービス時間**(すなわち, 単位(deterministic)分布に従う)の仮定をそれぞれ表示する.

さて, システムは時刻 $t = 0$ に動作を開始するものと仮定しよう. $X(t)$ $(t \geq 0)$ は時刻 $t$ におけるシステム内の(サービス中および待機中の)ジョブ数を表す. 遷移 $X(t) \to X(t')$ $(t' > t)$ は, 明らかに, ランダム到着と分布 $B(x)$ に従うというサービス時間の仮定によって導入される偶然性の効果によって決定付けられる. そうした理由から, $X(t)$ は**確率過程**(stochastic process)または**ランダム過程**(random process)といわれる. $X(t)$ で表されるシステムの状態はシステム内ジョブ数であるので, 状態空間は単に非負整数である $X(t)$ から成る. $X(t)$ は, 一般に, 初期値 $X(0)$, および全ての $t$ と $t'$ $(t' > t \geq 0)$ について遷移 $X(t) \to X(t')$ を支配する確率分布を指定したときに定義されることになる. これらの遷移は, 到着, サービス時間, およびサービス規律に関する仮定によって完全に決定される†. 前節で述べたように, われわれの関心は, $X$

† 当面の目標にとってサービス規律は何の影響も持たない. というのは, 順序付けは実行時間に独立に行われるものと仮定しているからである. (それにまた, 1個以上のジョブがシステム内にあるときはいつでもプロセッサはサービスに従事しているようにする規律だけに関心があるものと了解する.)

(0)を与えられ，したがって式(4.2.3)が成り立つことを仮定して，式(4.2.2)の確率分布を計算し，それにより $X(t)$ の特性を記述することにある．

$$(4.2.2)\qquad p_n(t) = \Pr[X(t) = n] \quad (t > 0,\ n = 0,\ 1,\ 2,\ \ldots)$$

$$(4.2.3)\qquad p_n(0) = \begin{cases} 1\,(X(0) = n) \\ 0\,(\text{その他のとき}) \end{cases}$$

$p_n(t)$ は次のように計算する．システムが状態 $X(t) \geq 1$ にあると仮定しよう．微小な時間間隔 $\Delta t$ に対する可能な遷移 $X(t) \to X(t + \Delta t)$ を評価したい．仮定に従えば，$\Delta t$ 内に一つの到着がある確率は $\lambda\Delta t + o(\Delta t)$ であり，到着がない確率は $1 - \lambda\Delta t + o(\Delta t)$，他のあらゆる到着事象は確率 $o(\Delta t)$ で起こる．$X(t) \geq 1$ であるので，同様に一つの退去がある確率と退去のない確率はそれぞれ $\mu\Delta t + o(\Delta t)$ と $1 - \mu\Delta t + o(\Delta t)$ であり，他の全ての退去事象は確率 $o(\Delta t)$ で生じる．そこで，到着時間間隔とサービス時間の独立性を用い，$\Delta t$ と同位の確率で生起する事象だけを考慮すると，($X(t) \geq 1$ に対して)次のように書くことができる．

$$\begin{aligned}
\Pr[X(t + \Delta t) = X(t) + 1] &= \text{P}[1\text{個の到着があり，退去がない}] \\
&= [\lambda\Delta t + o(\Delta t)]\,[1 - \mu\Delta t + o(\Delta t)] \\
&= \lambda\Delta t + o(\Delta t) \\
\Pr[X(t + \Delta t) = X(t)] &= \Pr[\text{到着がなく，退去もない}] \\
&= [1 - \lambda\Delta t + o(\Delta t)]\,[1 - \mu\Delta t + o(\Delta t)] \\
&= 1 - (\lambda + \mu)\Delta t + o(\Delta t) \\
\Pr[X(t + \Delta t) = X(t) - 1] &= \Pr[\text{到着はなく，1個の退去がある}] \\
&= [\mu\Delta t + o(\Delta t)]\,[1 - \lambda\Delta t + o(\Delta t)] \\
&= \mu\Delta t + o(\Delta t)
\end{aligned}$$

$X(t)$ に対する $X(t + \Delta t)$ の他の全ての可能性は確率 $o(\Delta t)$ を持つ．したがって，本節での記号を使えば，上記の方程式を集めて次のように式を立てることができる．

$$\begin{aligned}
p_n(t + \Delta t) = p_n(t)\,[1 - (\lambda + \mu)\Delta t + o(\Delta t)] &+ p_{n+1}(t)\,[\mu\Delta t + o(\Delta t)] \\
&+ p_{n-1}(t)\,[\lambda\Delta t + o(\Delta t)]
\end{aligned}$$

$\Delta t$ よりも低位の項を無視すると共に項のグループ分けを為直して $\Delta t$ で割れば次式のようになる．

$$\lim_{\Delta t\to 0}\frac{p_n(t+\Delta t)-p_n(t)}{\Delta t}=\mu p_{n+1}(t)-(\lambda+\mu)p_n(t)+\lambda p_{n-1}(t)$$

したがって，

$$p'_n(t)=\mu p_{n+1}(t)-(\lambda+\mu)p_n(t)+\lambda p_{n-1}(t) \tag{4.2.4}$$

今度は $X(t)=0$ と仮定しよう．この場合には退去の可能性は無視できるので次のように書ける．

$$p_0(t+\Delta t)=(1-\lambda\Delta t)p_0(t)+\mu\Delta t p_1(t)$$

これから次式を得る．

$$p'_0(t)=\mu p_1(t)-\lambda p_0(t) \tag{4.2.5}$$

与えられた初期条件(4.2.3)と制約条件 $\sum_{n=0}^{\infty}p_n(t)=1$ に従う式(4.2.4)および(4.2.5)の解は，時間に依存するシステムの振舞を記述して，その解は待ち行列理論の標準的教科書に見出すことができる†．しかし，この単純な待ち行列に対してさえも，どちらかといえば複雑で一見しただけでは解釈が困難である．こうした理由や，考察された他の多くのモデルに対してそのような解が見付かっていないといったもっと根本的な理由から，このような解によって記述される過渡的なあるいは時間に依存する振舞にこれ以上深入りするようなことはしないであろう．その代り，$t$ を大にしたときの確率 $p_n(t)$ の極限的振舞に対象を限定してそこに関心を注ぐことにする．

実際，確率 $p_n(t)$ がある極限値に近付くならば，この極限確率は**統計的平衡状態**(statistical equilibrium)(または**定常状態**(steady state))におけるシステムの振舞を記述しているといい，その**定常確率分布**(stationary probability distribution)は確率 $p_n=\lim_{t\to\infty}p_n(t)$ $(n=0, 1, 2, \ldots)$ によって与えられる．直観的には，ジョブの到着率 $\lambda$ が率 $\mu$——この率でジョブを実行することができる——より小であるという単純な条件の下で定常分布の存在を期待することができよう．このことは，実際，この分布が初期条件 $X(0)$ に依存しないという事実と共に証明することができる．しかしながら，ここではまず，定常分布が存在するという仮定に立って論を進め，その後に，この確率分布が $\lambda<\mu$ のときに限り満足に定義されるということを示そう．他のモデルについての解析も同じ方法で行う．

† 人口モデルへの応用により式(4.2.4)と(4.2.5)はしばしば出生死滅方程式(birth-and-death-equation)と呼ばれ，対応する過程は出生死滅過程といわれる[1]．

定常確率 $p_n$ を計算するため，定常状態では $\lim_{t\to\infty}p'_n(t)=0$ であるという事実を使うと，式(4.2.4)と(4.2.5)は次のように簡単な式となる．

$$\mu p_{n+1}-(\lambda+\mu)p_n+\lambda p_{n-1}=0 \quad (n\geq 1) \tag{4.2.6}$$

$$\mu p_1-\lambda p_0=0 \tag{4.2.7}$$

式(4.2.6)と(4.2.7)は，与えられた状態 $n\geq 1$ への遷移率(または，流量)$(\mu p_{n+1}+\lambda p_{n-1})$がその状態からの流量$(\lambda p_n+\mu p_n)$と平衡しているという定常状態**平衡**(balance)方程式の持つべき特徴を具えていることに注意しよう．$\rho=\lambda/\mu$ と定義して，式(4.2.6)と(4.2.7)を系統的に解いて行くと，次のようになることが分る．

$$p_n=\rho^n p_0 \quad (n=0, 1, 2, \ldots)$$

制約条件 $\rho<1$ および $\sum_{n=0}^{\infty}p_n=1$ を課すると，$p_0=1-\rho$ となり，それによって次の結果を得る．

$$p_n=(1-\rho)\rho^n \quad (n=0, 1, 2, \ldots) \tag{4.2.8}$$

$\rho=1-p_0$ はシステムが稼働中(busy)である平衡状態確率に他ならないことに注意しよう．一見して分るように，式(4.2.8)はパラメータ $\rho$ を持つ幾何分布であることを示しており，$\rho<1$ はこの分布が存在するための条件である．明らかに，$\rho<1$ すなわち $\lambda<\mu$ であるので，到着率は最大退去率よりも小でなければならない．こうした意味から，パラメータ $\rho$ はしばしば**利用率**(utilization factor)または**トラフィック密度**(traffic intensity)と呼ばれている．ここでは前者の言葉を用いる．

それでは，母関数の方法を用いて式(4.2.8)を導出しよう．$P(z)=\sum_{n=0}^{\infty}p_nz^n$ は $p_n$ に対する(確率)母関数を表す．式(4.2.6)に $z^n$ を乗じ，全ての $n\geq 1$ について総和を取って式(4.2.7)を加えると，次式のようになる．

$$\sum_{n=0}^{\infty}p_{n+1}z^n+\rho\sum_{n=1}^{\infty}p_{n-1}z^n=(1+\rho)\sum_{n=0}^{\infty}p_nz^n-p_0$$

$P(z)$を導入すると，次のようになる．

$$\frac{1}{z}[P(z)-p_0]+\rho zP(z)=(1+\rho)P(z)-p_0$$

これにより次式を得る．

$$P(z) = \frac{p_0}{1 - \rho z}$$

$\lim_{z\to 1} P(z) = 1$ を使うと $p_0 = 1 - \rho$ を得る．$P(z)$を $z$ の冪級数として表現すると次のようになる．

$$P(z) = (1-\rho) + (1-\rho)\rho z + \cdots + (1-\rho)\rho^n z^n + \cdots$$

この式の $n$ 番目の項の係数によって式(4.2.8)が与えられる．

多くの同様な問題では，$P(z)$を求めることはできるけれども，その式から具合よく直接に $p_n$ が得られるという訳ではないので，$P(z)$の計算の実際を，本問題で最後まで実行して見せることにより示した．$P(z)$が得られると，決まりきった流儀(補遺 A を見よ)で最初の二つのモーメントを計算することが可能になる．通常は，これがシステムの輻輳について知りたいことの大部分を占める．また，容易に $p_0 = \lim_{z\to 0} P(z)$を求めることもできるから，システムが稼働中である確率 $1 - p_0$ が得られる．

式(4.2.8)から統計的平衡状態におけるシステム内個数の平均と分散が次の通り与えられる．

(4.2.9)
$$\overline{n} = \frac{\rho}{1-\rho}$$

(4.2.10)
$$\mathrm{Var}(n) = \frac{\rho}{(1-\rho)^2}$$

最後に一言付け加えると，上述した結果は，先験的に既知であると仮定される到着ジョブの実行時間に関する情報に基づいてジョブを順序付けるようなことはしない全てのサービス規律に対して適用可であるということを強調しておこう．さて，次に，もっと一般的な M/G/1 型システムを調べ，その後で，FIFO システムに対する待ち時間の分布を導出することにする．

### 4.2.2 M/G/1 型待ち行列システム

到着時間間隔とサービス時間が各々独立な指数分布に従っているということが仮定された場合は，過程 $X(t)$ は次のような極めて重要な性質を持つことになる．すなわち，任意に与えられた時点 $t_0$ から観た $X(t)$の将来の振舞は，現在の状態 $X(t_0)$にのみ依存して，$t_0$ に先立つ過去には依存しないという性質である．非形式的に述べると，この**無記憶性**(マルコフ性)が**マルコフ過程**(Markov process)を定義し，それが解析に実質的な簡易性を齎している．以下に

述べる待ち行列モデルでは，到着過程がこの性質を保持しているとはいえ，サービス時間については一般分布が仮定されているので，この場合に対し，$X(t)$の解析がどのように行われるかが関心の的である．このような仮定の下では，ある時点 $t_0$ において，あるジョブが実行の途中にあるとすると，$X(t)$ $(t > t_0)$の振舞は，$X(t_0)$に依存するだけでなく，実行途中のジョブが既に受けたサービスの量にも依存することになる．それゆえ，$X(t)$は最早マルコフ過程ではない．上述の性質によって示唆される前出の接近法の一つの拡張を次の素描によって示すことができる．

時刻 $t$ において実行中のジョブ(もし存在するならば)によって既に受取られたサービスの量を示す補助変数 $x$ の値を状態空間が含むように $X(t)$の定義を為直すことができる．明確に述べるならば，新過程 $X'(t)$ の値は対$(n, x)$である．ただし，$n$ は時刻 $t$ におけるシステム内(ジョブの)個数を表し，$x$ は上記の定義の通りである．新過程は今やマルコフ性を持つので，その基盤の上に立って解析を進めることが可能である．しかし，ここでは，この接近法による困難な解析を避けて，限られた目的に取り遙かに簡単な別の方法を考えることにする．

これから記述する技法は，全ての時点で $X(t)$がマルコフ性を持つということはいえないにもかかわらず，隠れている(embedded)時点の系列が数多存在していて，その系列上では $X(t)$はマルコフ性を持つという事実の認識がその動機になっている．そのような系列の例として，サービス分布 $B(x)$を持つ M/G/1 型システムでは，$i$ 番目のジョブの実行終了直後の瞬間を $t_i$ とする時点の系列 $t_1, t_2, \ldots$を定義することができる．

サービス時間と到着時間間隔が互いに独立であるがゆえに，$X(t)$は時点 $t_i$ $(i = 1, 2, \ldots)$ 上でマルコフ性を持つ．この事実から，$t_i$ を**再生点**(regeneration point)ともいうが，$X(t)$に関する**隠れマルコフ連鎖**(embedded Markov chain)と呼ばれるマルコフ連鎖が離散時間過程 $X(t_i)$ $(i = 1, 2, \ldots)$によって構成されることが分る．記法を簡単にするため，$X_i \equiv X(t_i)$と置いて，このマルコフ連鎖を$\{X_i\}$で表すことにしよう．

マルコフ連鎖$\{X_i\}$は，**1段遷移確率**(one-step transition probability)

$$\pi_{ij} = \Pr\{X_{k+1} = j | X_k = i\} \quad (k = 1, 2, \ldots) \tag{4.2.11}$$

が時間パラメータ $k$ の関数ではないという意味で**一様**(homogeneous)である．$\{X_i\}$の定常分布$\{\pi_i\}$は，連立**平衡方程式**(equilibrium equation)†

† $\{X_i\}$に対して定常分布が存在するための一般的条件が補遺 A で与えられる．これらの条件は本書で議論される応用に対して常に成立する．

(4.2.12) $$\pi_j = \sum_{i=0}^{\infty} \pi_{ij}\pi_i \quad (j = 0, 1, 2, \ldots)$$

の解であって，しかも正規化条件 $\sum_{j=0}^{\infty} \pi_j = 1$ を満足するものとして定義される．さて，ポアソン到着と一般サービス時間分布の仮定の下で，式(4.2.12)の確率に対する母関数を計算しよう．なお，演習問題 4.10 では定常分布のモーメントを見出すもっと直接的な技法を示す．

式(4.2.12)に $z^j$ を乗じ，全ての $j \geq 0$ について総和を取ることから始める．和を取る順序を変えると下記の母関数が得られる．

(4.2.13) $$P(z) = \sum_{j=0}^{\infty} \pi_j z^j = \sum_{j=0}^{\infty} z^j \sum_{i=0}^{\infty} \pi_i \pi_{ij} = \sum_{i=0}^{\infty} \pi_i \sum_{j=0}^{\infty} \pi_{ij} z^j$$

次に，遷移確率 $\pi_{ij}$ を計算しよう．

$X_k = n > 0$, すなわち，システム内に少なくとも 1 個のジョブがあり，$(t_k, t_{k+1})$が 1 サービス期間に対応していると仮定しよう．$(t_k, t_{k+1})$間に実行するジョブは退去することになる唯一のジョブであるから，$X_{k+1} \geq n - 1$ でなければならない．このことから，$j < i - 1$ に対しては $\pi_{ij} = 0$ となり，$X_{k+1} - X_k + 1$ は$(t_k, t_{k+1})$内に生じる到着数を表すことになる．したがって，$q_m (m = 0, 1, 2, \ldots)$で 1 サービス期間内に $m$ 個の到着がある確率を示すならば，$i > 0$ および $j \geq i - 1$ に対して $\pi_{ij} = q_{j-i+1}$ となる．

今度は，$X_k = 0$, すなわち，$t_k$ は**遊休期間**(idle period)の始まりであると仮定しよう．この遊休期間の長さに関係なく，$X_{k+1} = j$ となる確率は単に遊休期間を終らせる到着ジョブの実行中に $j$ 個の到着がある確率である．したがって，$j \geq 0$ に対して $\pi_{0j} = \pi_{1j} = q_j$ となる．ここで式(4.2.13)の $\pi_{ij}$ に代入を行うと，総和の第 1 項を分離して書くならば次のようになる．

(4.2.14) $$P(z) = \pi_0 \sum_{j=0}^{\infty} q_j z^j + \sum_{i=1}^{\infty} \pi_i \sum_{j=i-1}^{\infty} q_{j-i+1} z^j$$

$A(z) = \sum_{m=0}^{\infty} q_m z^m$ で 1 サービス期間内到着数の分布に対する母関数を表すと，式(4.2.14)は次のように簡略な形で表すことができる．

(4.2.15) $$P(z) = \pi_0 A(z) + \frac{A(z)}{z}[P(z) - \pi_0]$$

$P(z)$について解けば，次式が得られる．

(4.2.16) $$P(z) = \pi_0 \frac{(1-z)A(z)}{A(z)-z}$$

さらに論を進めるためには $A(z)$を調べる必要がある．今，$f_m(t)$は，式(4.1.9)で与えられるように時間 $t$ 内に $m$ 個のポアソン到着が生じる確率を示す．したがって，サービス時間分布の上でこの確率の平均を取ると，次のようになる．

$$q_m = \int_0^\infty f_m(t)dB(t) \quad (m = 0, 1, 2, \ldots)$$

式(4.1.9)を代入し，母関数を計算すると，次のようになる．

$$A(z) = \sum_{m=0}^{\infty} z^m \int_0^\infty \frac{(\lambda t)^m}{m!} e^{-\lambda t} dB(t)$$

総和と積分の順序を逆にすると，次式を得る．

(4.2.17) $$A(z) = \int_0^\infty e^{-\lambda t(1-z)} dB(t)$$

サービス時間のラプラス変換 $B^*(s) = \mathrm{E}[e^{-sS}]$を用いると，次式が得られる．

(4.2.18) $$A(z) = B^*(\lambda - \lambda z)$$

式(4.2.17)を用いれば，式(4.2.16)に戻って $\pi_0$ と分布$\{\pi_i\}$のモーメントを計算することができる．$\lim_{z\to1} P(z) = 1$ を使い，式(4.2.16)の分子と分母が共に $z = 1$ で零になることに注意すれば，**ロスピタル**(l'Hospital)**の公式**を応用して次式が得られる．

$$\frac{-\pi_0 A(1)}{A'(1)-1} = 1$$

ところで，式(4.2.17)から $A(1) = 1$ である．式(4.2.17)を微分すると $A'(1) = \rho$ となることが分り，したがって次式が得られる．

(4.2.19) $$\pi_0 = 1 - \rho$$

ここで $\rho = \lambda\mathrm{E}(S)$ は利用率である．(明らかに，$0 \le \rho < 1$ は統計的平衡にとって必要である．)こうして$\{X_i\}$の定常分布が母関数

$$P(z)=\frac{(1-\rho)(1-z)B^*(\lambda-\lambda z)}{B^*(\lambda-\lambda z)-z} \tag{4.2.20}$$

によって定義される．１次モーメントを求めるため，再びロスピタルの公式を使えば，次のようになる†.

$$\overline{n}=\lim_{z\to 1}\frac{dP(z)}{dz}=\rho+\frac{A''(1)}{2(1-\rho)}$$

式(4.2.17)から $A''(1)=\lambda^2\mathrm{E}(S^2)$ が得られ††，したがって次のようになる．

$$\overline{n}=\rho+\frac{\lambda^2\mathrm{E}(S^2)}{2(1-\rho)} \tag{4.2.21}$$

次式

$$C^2(S)=\frac{\mathrm{Var}(S)}{\mathrm{E}^2(S)}=\frac{\mathrm{E}(S^2)}{\mathrm{E}^2(S)}-1 \tag{4.2.22}$$

で定義される**変動係数**(coefficient of variation)$C(S)$を導入すると，式(4.2.

---

† 訳注

$$\begin{aligned}
n &= \lim_{z\to 1}\frac{dP(z)}{dz}\\
&= \lim_{z\to 1}\frac{d}{dz}\left\{\pi_0\frac{(1-z)A(z)}{A(z)-z}\right\} \quad [\text{式}(4.2.16)]\\
&= \lim_{z\to 1}\pi_0\frac{-A^2(z)+A(z)-z(1-z)A'(z)}{(A(z)-z)^2}\\
&= \lim_{z\to 1}\pi_0\left\{\frac{A'(z)}{1-A'(z)}-\frac{z(1-z)A''(z)}{2(A(z)-z)(A'(z)-1)}\right\} \quad [\text{分母子をそれぞれ微分する}]\\
&= \rho+\lim_{z\to 1}\frac{z(1-z)A''(z)}{2(A(z)-z)} \quad [\pi_0=1-\rho,\ A'(1)=\rho\ \text{を用いる}]\\
&= \rho+\lim_{z\to 1}\frac{(1-2z)A''(z)+z(1-z)A'''(z)}{2(A'(z)-1)} \quad [\text{さらに分母子をそれぞれ微分する}]\\
&= \rho+\frac{A''(1)}{2(1-\rho)}
\end{aligned}$$

†† 訳注

$$\begin{aligned}
A''(1) &= \lim_{z\to 1}\frac{d^2A(z)}{dz^2}\\
&= \lim_{z\to 1}\frac{d^2}{dz^2}\int_0^\infty e^{-\lambda t(1-z)}dB(t)\\
&= \lim_{z\to 1}\int_0^\infty (\lambda t)^2e^{-\lambda t(1-z)}dB(t)\\
&= \lambda^2\int_0^\infty t^2dB(t)\\
&= \lambda^2\mathrm{E}(S^2)
\end{aligned}$$

21)をもっと普通に用いられる形に書き直すことができる．すなわち，

$$\overline{n} = \rho + \frac{\rho^2[1 + C^2(S)]}{2(1-\rho)} \tag{4.2.23}$$

分布$\{\pi_j\}$の高次のモーメントも同様なやり方で計算することができる．

$\{X_i\}$の定義に従えば，式(4.2.23)はジョブの退去によってシステム内に残されるジョブの平均数を表す．システムの状態は退去直後の瞬時においてだけ観察されているので，一般に，分布$\{\pi_j\}_{j=0}^{\infty}$は連続な時間パラメータを持つ過程$X(t)$に対する定常分布とは異なるということを予期するのが合理的であろう．つまり，システムの評価というのは一般にそれを観察するために選んだ時点に依存して変る(あるいは偏る)ものである．しかし事実はこれら二つの分布が結局は寸分違わず同じものであるということが判明する．一般に，この性質は任意の時点で可能な状態変化が$+1$(到着)と$-1$(退去)だけであるような任意の待ち行列システムに適用される．この性質にぴったり合致して指数サービス時間分布に対する式(4.2.9)の結果が，$\mathrm{E}(S^2) = 2/\mu^2$を式(4.2.21)に代入することにより見出される．

$\overline{n}$が平均到着時間間隔と平均サービス時間によるだけでなくサービス時間分布の2次モーメントによっても決定されるということに特に注意しよう．式(4.2.21)によって示されるように，$\overline{n}$はサービス時間の分散に対して直線的に増加する．たとえば，一定サービス時間を考えると，サービス時間の変動は零であるので次のようになる．

$$\overline{n} = \rho + \frac{\rho^2}{2(1-\rho)} = \frac{\rho(1-\rho/2)}{1-\rho} \tag{4.2.24}$$

これは指数サービス時間(その変動係数は1である)に対する式$\rho/(1-\rho)$と比較されるべきである．微小な$\rho$に対しては平均の差は微小であるが，$\rho$が1(飽和(saturation)条件)に近付くにつれて両者の比がほとんど1対2になるくらいの差が出る．

システムが稼働中である確率$1-\pi_0=\rho$をプロセッサ内ジョブ数の平均と解釈すると，式(4.2.23)の第2項は待ち行列内で待っている平均個数$\overline{n}_q$になると予想されるであろう．直接計算することによって示されるように，正しくこれは成り立つ．すなわち，

$$\overline{n}_q = \frac{\rho^2[1 + C^2(S)]}{2(1-\rho)} \tag{4.2.25}$$

さらに，一般サービス時間に対する上記の結果は，ジョブの実行時間に関する情報を利用しないような横取り割込み不可の(待ち行列からの)ジョブ選択規則に対しても適用可能である.

### 4.2.3 待ち時間

**4.2.3.1 残存待ち時間**(Residual Waiting Time) FIFO システムに対する待ち時間を計算する前に，後の議論で横取り割込み不可規律に対する待ち時間の期待値を導出するときにその解が役に立つもっと単純な問題を扱うことにする.

統計的平衡状態における M/G/1 型システムへのランダム到着を考えよう．到着時にシステムが稼働中であるとして，実行の途中にあるジョブの残り実行時間の(条件付)確率分布を $R(x)$ で表す．問題は，残存実行時間の期待値 $R$ を見出すことである．再生理論では $R$ は**平均前進再帰時間**(mean forward recurrence time)といい, 信頼性理論では**平均余生時間**(mean residual lifetime)という．下記では，$R$ の簡潔でかつ形式的でない導出を与えよう．$R(x)$を導出するまでの完全な解析は補遺 B で与える．(演習問題 4.12 をも見よ.)

単純に $R$ は $\mathrm{E}(S)/2$ であるとまずは言いたくなるであろう．しかし，一般サービス時間分布に対してそれが誤りであることは，ランダムな到着は短いジョブより長いジョブの実行中にこそ起こりそうであるということによって説明される．次の議論ではこの着眼を活用する．$b(x)$はサービス時間の密度関数，$b'(x)$はランダム到着が起こるサービス期間の長さの密度関数を表すものとする．$b(x)dx$ は長さ $x$ のサービス期間の相対頻度と解釈することができるから，密度 $b'(x)$は $x$ と $b(x)$に比例するはずである．$K$ を比例定数とすると，$Kxb(x)$が確率密度であるためには $K = 1/\mathrm{E}(S)$ でなければならないということが分る．このことから次式を得る.

$$b'(x) = \frac{xb(x)}{\mathrm{E}(S)} \tag{4.2.26}$$

ところで，長さ $x$ のサービス期間内にランダム到着があったとすると，残存実行時間の条件付期待値は単に $x/2$ である．したがって，次式が成り立つ.

$$R = \int_0^\infty \left(\frac{x}{2}\right) \frac{xb(x)}{\mathrm{E}(S)} dx = \frac{\mathrm{E}(S^2)}{2\,\mathrm{E}(S)} \tag{4.2.27}$$

後で，この結果をこれとは独立した議論で確かめる機会がやってこよう．$R$

はサービス時間における変動に対して直線的に増加し，サービス時間が一定のときに限り最小値 $E(S)/2$ に等しくなることに注意しよう．また，指数サービス時間に対しては，式(4.2.27)は $E(S)$になってしまい，無記憶性に整合することが分る．

今度は，到着時にシステムが稼働中であるという条件を取除こう．別の言い方をすると，到着時刻後，行列から次に選び出されるジョブにプロセッサを割当てる時点までに経過する時間の平均を求めることにする．この量を $E(S_r)$で表す．システム†が空であるときの到着ジョブは到着と同時にプロセッサを使用し始めることができるということと，システムが稼働中である確率は $\rho$ であるということから，統計的平衡状態において次式が成立する．

$$E(S_r) = \rho R$$

$R$ に代入を行い，$\rho = \lambda E(S)$ と置けば，次式を得る．

$$E(S_r) = \frac{\lambda E(S^2)}{2} \tag{4.2.28}$$

この結果は，到着ジョブがそれぞれ異なるサービス時間分布を持つクラスにグループ分けされ得るシステムに対しても容易に拡張される．特に，もし $\lambda_p$ と $B_p(x)\,(p = 1, 2, \ldots)$がそれぞれ到着率とサービス時間分布を表すならば，クラス $p$ のジョブに対して，同じ議論を適用することにより統計的平衡状態の仮定の下で次式を得る††．

† **ポアソン到着を伴う**待ち行列の重要な性質を暗黙に使用しているし，また今後も使用し続ける．特に，平衡状態行列長分布は統計的平衡状態における到着によって見出される分布に一致するということが示し得る．直観的には，この結果はポアソン到着の‘ランダム性’から来る．（区別のため前者の分布は‘外部観測者’分布ともいわれる）．

†† **訳注**　この場合は，サービス時間の密度あるいはサービス期間の相対頻度が

$$b(x) = \frac{\sum_{p\geq 1}\lambda_p b_p(x)}{\sum_{p\geq 1}\lambda_p}$$

となり，平均は次のようになる．

$$E(S) = \frac{\sum_{p\geq 1}\lambda_p E(S_p)}{\sum_{p\geq 1}\lambda_p}$$

ランダム到着が生じるサービス期間の長さの密度関数はやはり $x$ と $b(x)$に比例し，$K$を比例定数とすれば次のようになる．

$$b'(x) = Kxb(x)$$

$b'(x)$が確率密度であるためには

$$1 = \int_0^\infty b'(x)\,dx = KE(S) \quad \nearrow$$

$$\text{(4.2.29)} \qquad \mathrm{E}(S_r) = \frac{1}{2}\sum_{p\ge 1}\lambda_p \mathrm{E}(S_p{}^2)$$

なお，4.5節での優先権待ち行列の解析においては式(4.2.29)を使用するはずである．

**4.2.3.2 FIFO 待ち時間分布**(FIFO Waiting Time Distribution) FIFOシステムに対しては平衡状態待ち時間分布 $W(x)$のラプラス変換がシステム内個数の母関数 $P(z)$から容易に得られる．$\{\pi_i\}$は退去するジョブによって背後に残されるジョブの個数の確率塊関数を与えるものであったということを想起しよう．背後に残されるジョブというのは，見方を変えるならば，正に退去するジョブがシステム内に滞留している間に到着したものに他ならない．ところで，式(4.2.18)，すなわち，$A(z) = B^*(\lambda - \lambda z)$はサービス期間内に生じる到着の個数を記述する式であった．そこで，それにならえば，システム内待ちの時間内に生じる到着の個数を記述する式は次式のように表されることになる．

$$\text{(4.2.30)} \qquad P(z) = W^*(\lambda - \lambda z)$$

式(4.2.20)の左辺に式(4.2.30)を代入し，さらに変数変換 $s = \lambda - \lambda z$ を行えば，次式が得られる．

$$\text{(4.2.31)} \qquad W^*(s) = \frac{(1-\rho)sB^*(s)}{s - \lambda + \lambda B^*(s)}$$

---

↘ から $K = 1/\mathrm{E}(S)$でなければならない．長さ $x$ のサービス期間内にランダム到着があれば，残り実行時間の条件付期待値は $x/2$ であるから

$$R = \int_0^\infty \left(\frac{x}{2}\right)\frac{xb(x)}{\mathrm{E}(S)}dx = \frac{\sum_{p\ge 1}\lambda_p \mathrm{E}(S_p{}^2)}{2\sum_{p\ge 1}\lambda_p \mathrm{E}(S_p)}$$

到着時にシステムが稼動中であるという条件を取り除くと，

$$\mathrm{E}(S_r) = \rho R$$

ところで，

$$\rho = \left(\sum_{p\ge 1}\lambda_p\right)\cdot \mathrm{E}(S) = \sum_{p\ge 1}\lambda_p \mathrm{E}(S_p)$$

となるから，

$$\mathrm{E}(S_r) = \frac{1}{2}\sum_{p\ge 1}\lambda_p \mathrm{E}(S_p{}^2)$$

を得る．

システム内待ち時間は列内時間(行列内待ち時間)とサービス期間の和である．したがって，**列内時間**(queueing time)分布 $V(x)$ のラプラス変換は式 $W^*(s) = B^*(s)V^*(s)$ が成り立つことを利用して次式の通り求められる．

$$V^*(s) = \frac{(1-\rho)s}{s-\lambda+\lambda B^*(s)} \tag{4.2.32}$$

1次モーメントを $W$ および $V$ で表すならば，式(4.2.31)と(4.2.32)をそれぞれ微分することにより次の通りの結果が得られる．

$$W = \mathrm{E}(S)\left[1 + \frac{\rho[1+C^2(S)]}{2(1-\rho)}\right] \tag{4.2.33}$$

および

$$V = \mathrm{E}(S)\frac{\rho[1+C^2(S)]}{2(1-\rho)} \tag{4.2.34}$$

式(4.2.33)，または(4.2.34)）はしばしば**ポラチェック-ヒンチン**(Pollaczek-Khintchine)の公式と呼ばれる．

これとは独立な次の議論によりM/M/1型システムに対して式(4.2.31)を確かめることができる．到着時のシステム内個数 $n$ によって条件付けられた待ち時間の平衡分布を $W_n(x)$ で表そう．無記憶性ゆえに，到着時に実行中のジョブの残り実行時間は行列内のジョブに対するのと同じ指数分布を持つ．したがって，FIFO規則では，$W_n(x)$ は $n+1$ 個の互いに等しくかつ独立な指数確率変数の和の分布に従うことになる．これは補遺Aで述べられるガンマ型(または特殊アーラン型)の密度に対応する．$\mu$ をサービス時間分布のパラメータとすると，次式が成り立つ．

$$W^*_n(s) = \left(\frac{\mu}{\mu+s}\right)^{n+1} \tag{4.2.35}$$

式(4.2.8)を用いて条件を取扱うと，式(4.2.8)が定常幾何分布であることから，条件付きでない分布が下記の通り求められる．

$$W^*(s) = \sum_{n=0}^{\infty} p_n W^*_n(s) = (1-\rho)\sum_{n=0}^{\infty}\rho^n\left(\frac{\mu}{\mu+s}\right)^{n+1} \tag{4.2.36}$$

すなわち，

(4.2.37) $$W^*(s) = \frac{(1-\rho)\mu}{(1-\rho)\mu + s}$$

この式は，下記の指数分布のラプラス変換であることがみてとれよう．

(4.2.38) $$W(x) = 1 - e^{-(1-\rho)\mu x} (x \geq 0)$$

一方，式(4.2.37)は，指数サービス分布のラプラス変換を式(4.2.31)に代入することによっても同様に得られることに注意するべきである．なお，平均値 $W$ は次の通りである．

(4.2.39) $$W = \frac{1/\mu}{1-\rho}$$

したがって，システム内における待ち時間もまたマルコフ性を持つ．それぞれが同一の指数分布を持つ独立な確率変数の‘幾何分布に従う和’は，それ自身指数分布に従う† という式(4.2.38)に示される事実は注目に値する．離散型の場合は，もし個々の確率変数が幾何分布に従うならば，対応する分布は幾何分布となる．

到着時のシステム内個数 $n \geq 1$ が与えられたときの条件付列内時間分布 $V_n(x)$ は明らかに $W_{n-1}(x)$ である．空なシステムへの到着は列内で待つことがないから，列内時間の分布は次式のようになる．

$$V(x) = p_0 + \sum_{n=1}^{\infty} p_n W_{n-1}(x)$$

$W_{n-1}(x)$ のラプラス変換を使い，さらに式(4.2.8)を用いて条件付けを取払うと，次式が容易に得られる．

(4.2.40) $$V(x) = 1 - \rho e^{-(1-\rho)\mu x}$$

この式は式(4.2.32)と整合している．$V(x)$ は $x = 0$ に集中している確率塊$(1-\rho)$との混合分布であることに注意しよう．

式(4.2.33)と(4.2.34)における平均値の表現に直接に到達することも可能である．たとえば，$V$ は，ランダムに選ばれた到着時刻 $t$ から次に実行すべきジ

† **訳注** 同一の指数分布に従う独立な確率変数を $X_1, X_2, \ldots$ と置く．$N$ は幾何分布 $\gamma$ に従う離散的確率変数であるとして，$X_1, X_2, \ldots$ の幾何分布 $\gamma$ に従う和を $S = X_1 + X_2 + \cdots + X_N$ と定義する．

ョブにとってプロセッサが使用可能になる時刻までの時間の期待値と，時刻 $t$ に列内で待っていたジョブの平均数 $\overline{n}_q$ にサービス時間の期待値を乗じたものとの和として表現することができる．明確に書くと次のようになる．

$$V = \overline{n}_q \mathrm{E}(S) + \mathrm{E}(S_r) \tag{4.2.41}$$

式(4.2.25)と(4.2.28)をこの式に代入すれば式(4.2.34)を得る†.

**4.2.3.3　リトルの公式**(Little's Result)　式(4.2.30)を微分し，極限 $z \to 1$ を取ると，リトルの公式を得る††．すなわち，

$$\overline{n} = \lambda W \tag{4.2.42}$$

これは，平衡状態におけるシステム内の平均個数が到着率とシステム内平均滞留時間の積に等しいということを述べている．また，この結果は，新しい到着が出会うシステム内の平均数($\overline{n}$)はそれが退去にあたって背後に残す平均数($\lambda W$)に等しいという保存原理を含んでいる．式(4.2.42)が成立するかどうかは，これまでに論じてきた文脈の範囲をはるかに超えて，本質的には，長時間走行における到着率が長時間走行における退去率に等しい定常状態の存在にのみ依存している．特に，この結果は，スケジューリング規則または到着過程のどんな個々の性質にも依存しない．(3.8節をも見よ．そこでは，リトルの公式が決定性順序付け問題に対して応用されている．)これらの性質の詳細な証明は Little [9] によって与えられている．

行列内における個数に限定するとリトルの公式は次のようになる．

---

† $V$ については式(4.2.34)を，$n_q$ については式(4.2.25)を式(4.2.41)に代入すれば，式(4.2.28)で $\mathrm{E}(S_r)$ について得た表現を独立に確かめることができるということに注意しよう．

†† **訳注**　式(4.2.30)を微分すると，

$$\frac{dP(z)}{dz} = [W^*(\lambda - \lambda z)]'(-\lambda)$$

極限 $z \to 1$ を取ると，左辺は

$$\lim_{z-1} \frac{dP(z)}{dz} = \overline{n}$$

右辺は

$$\lim_{z\to 1}[W^*(\lambda - \lambda z)]'(-\lambda) = (-\lambda)\lim_{s\to 0}[W^*(s)]' = \lambda W$$

となるから，リトルの公式

$$\overline{n} = \lambda W$$

を得る．

$$\overline{n}_q = \lambda V \tag{4.2.43}$$

興味深いことに，式(4.2.41)における $\overline{n}_q$に式(4.2.43)を代入すれば，$\overline{n}_q$[または $P(z)$]をあらかじめ知っておかなくても，$V$ については解くことができたのであるということに気付く．最後に，優先権のある待ち行列を研究するときにリトルの公式を個々の優先権クラスへ適用する機会があるということを注意しておこう．そこでは，たとえば，

$$\overline{n}_k = \lambda_k W_k (k = 1, 2, \ldots) \tag{4.2.44}$$

は，各 $k$ に対し優先権 $k$ のジョブの平均個数はその到着率をその平均待ち時間に乗じたものに等しいということを述べている．

### 4.2.4 全稼働期間分布

統計的平衡状態における M/G/1 型待ち行列の**全稼働期間**(busy period)の長さを表す確率変数を $D$ とする． 全稼働期間の分布 $H(y) = \Pr[D \le y]$を求めよう． 全稼働期間を開始させるジョブ $J_0$ が実行に $x$ 時間単位を必要とし，しかもその実行中に $n$ 個の到着ジョブ $J_1, J_2, .., J_n$ があるものと仮定する．ところで， 分布 $H(y)$は，$J_1, J_2, \ldots, J_n$ および続いて到着するジョブの実行順序からは影響を受けない．そこで，次に述べる規律を採用することにする．$J_0$ の完了後， $J_1$ を実行し， さらにシステム内に残留するジョブが $J_2, \ldots, J_n$ だけになるまで後続の到着ジョブを実行する．その時点で，再び同様のやり方で $J_3, \ldots, J_n$ だけが残りのジョブになるまで $J_2$ と後続の到着ジョブの実行を進める．この過程は， $J_n$ を実行し， 引き続いて後続の到着ジョブをも実行して，最終的にシステムが再度空になり，全稼働期間が終結するまで繰り返される．

$D_i$ は $J_i$ の実行開始から $J_{i+1}$ の実行開始までの経過時間を表すものとする$(i < n)$. $D_n$ は $J_n$ の実行開始から全稼働期間の終了までの時間間隔を表す．$X$ と $N$ は，それぞれ $J_0$ の実行時間とその間の到着数を表す確率変数であるとする．多少なりとも考えれば，$D_i (1 \le i \le n)$は互いに独立な確率変数であって，各々が全稼働期間と同じ分布を持っているという解決の鍵になる見解に到達するはずである． こうして， $J_0$ が $x$ 時間単位の実行時間を必要とし，しかもこの $x$ 時間単位の間に $n$ 個の到着が生じるという条件が与えられたとき，この条件付全稼働期間分布は次のラプラス変換を持つということが了解される．

$$\mathrm{E}(e^{-sD}|X=x,\, N=n) = e^{-sx}\mathrm{E}(e^{-s\Sigma_{i=1}^{n}D_i})$$
$$= e^{-sx}[H^*(s)]^n$$

$x$ の間のポアソン到着の個数 $n$ と $J_0$ のサービス時間 $x$ に関する条件付けを取除くと，次式を得る．

$$\mathrm{E}(e^{-sD}) = \int_0^\infty e^{-sx}\sum_{n=0}^{\infty}\frac{(\lambda x)^n}{n!}[H^*(s)]^n e^{-\lambda x}dB(x)$$

あるいは

(4.2.45) $$H^*(s) = \int_0^\infty e^{-sx}e^{-\lambda x[1-H^*(s)]}dB(x)$$

したがって，最終的には次式を得る．

(4.2.46) $$H^*(s) = B^*(s+\lambda-\lambda H^*(s))$$

この関数方程式は通常 $H^*(s)$ の解を陽関数の形で与えるように解くことができないにもかかわらず，モーメントは容易に導出できる(演習問題**4.1**)．なお，上記の解析法は，LIFO 規律に対する待ち時間分布を得るための演習問題**4.13**においても利用される．

## 4.3 ポアソン待ち行列における状態依存型の到着とサービス時間

平衡方程式(**4.2.6**)および(**4.2.7**)は，サービス規律がサービス時間によって影響を受けない基本的ポアソン待ち行列の有用かつ興味ある幾多の変種にも適応するように容易に一般化することができる．特に，到着率および(最大)サービス率がシステム状態(すなわち，システム内の個数)の関数であることを仮定しよう．$\lambda_n$ と $\mu_n$ は，待機中あるいは実行中の $n$ 個のジョブをシステム内に含むときの到着率とサービス率を表すものとする．式(**4.2.4**)および(**4.2.5**)を求め直すと下記の式を得る．

(4.3.1) $$p'_n(t) = \mu_{n+1}p_{n+1}(t) - (\lambda_n+\mu_n)p_n(t) + \lambda_{n-1}p_{n-1}(t) \quad (n \geq 1)$$

(4.3.2) $$p'_0(t) = \mu_1 p_1(t) - \lambda_0 p_0(t)$$

対応する平衡方程式は次のようになる．

(4.3.3) $$\mu_{n+1}p_{n+1} - (\lambda_n+\mu_n)p_n + \lambda_{n-1}p_{n-1} = 0$$

$$\mu_1 p_1 - \lambda_0 p_0 = 0 \tag{4.3.4}$$

式(4.3.3)と(4.3.4)を系統的に解いて行くと，次のようになる．

$$p_1 = \frac{\lambda_0}{\mu_1} p_0, \quad p_2 = \frac{\lambda_1}{\mu_2}\frac{\lambda_0}{\mu_1} p_0, \quad p_3 = \frac{\lambda_2}{\mu_3}\frac{\lambda_1}{\mu_2}\frac{\lambda_0}{\mu_1} p_0, \quad \cdots$$

あるいは

$$p_n = p_0 \prod_{i=}^{n} \frac{\lambda_{i-1}}{\mu_i} \quad (n = 1, 2, \ldots) \tag{4.3.5}$$

確率の総和は1に等しいと置くことにより $p_0$ を得る．すなわち，

$$p_0 = \left\{ 1 + \sum_{n=1}^{\infty} \prod_{i=1}^{n} \frac{\lambda_{i-1}}{\mu_i} \right\}^{-1} \tag{4.3.6}$$

以下では，計算機への応用において意味のあるいくつかの例題を解くことにする．

計算機の操作において発生する多くの待ち合わせ過程では，無制限に大きい待ち合わせ設備または記憶手段があるという仮定はそう強くは支持できない．指数分布に従うという仮定の下でこのようなシステムをモデル化するため，単一サーバ・システムにおいて最大限 $L$ 個のジョブ(プロセッサ内の1個を含む)が収容できると仮定しよう． システムが一杯($n = L$)であるのを見出す到着ジョブは再度立戻ることなく去るものと仮定する．次の条件を指定することにより式(4.3.5)と(4.3.6)の結果が使用できる．

$$\lambda_n = \begin{cases} \lambda & (n < L) \\ 0 & (n \geq L) \end{cases} \tag{4.3.7}$$
$$\mu_n = \mu \quad (n = 1, 2, \ldots, L)$$

これを式(4.3.5)と(4.3.6)に代入し $\rho = \lambda/\mu$ を用いると，次のようになる．

$$p_n = \frac{\rho^n (1-\rho)}{1-\rho^{L+1}} \quad (n = 0, 1, \ldots, L) \tag{4.3.8}$$

この式から， モーメントと待ち時間が通常の方法で見出される． 式(4.3.8)は全ての $\rho \geq 0$ に対して成立することに注意しよう．[$\rho = 1$ のとき式(4.3.8)は一様分布 $p_n = 1/(L+1)$ となる.] $\lambda$ と $\mu$ の相対的な値がどうであろうとシステム内の個数は $L$ で押さえられているので， $p_n$ は全ての $n (0 \leq n \leq L)$ およ

び $0 < \rho < \infty$ に対して非零となる.

多くの計算機モデルに対するもう一つの重要な制限条件は, **有限入力源**(finite-source)の仮定によって表現されるものである. タイムシェアリングへの応用においては特に, 利用者(たとえば, コンソール)の数があまりに小であるかもしれないので, 基本的なポアソン到着機構に本来付き物の無限入力源(infinite-source)の仮定は, 現実の到着過程の表現としては貧弱なものになってしまう可能性があろう.

(たとえば, 遠隔ジョブ入力システム(remote job entry system)において)$N$ 個の端末が存在するものとし, 一つの端末においてある一つのジョブの入力完了時刻から次のジョブの入力完了時刻までの経過時間はパラメータ $\lambda$ の指数分布に従うと仮定しよう. 前と同様に指数分布するサービス時間を仮定すると, 次のようにして式(4.3.5)と(4.3.6)の結果を用いることができる. システム内に $n$ 個のジョブが存在するときは, 新しいジョブを入力するために $N-n$ 個の端末が使用可能であるから, $\lambda_n$, $\mu_n$ を次のように置くことができる.

(4.3.9)
$$\lambda_n = \begin{cases} (N-n)\lambda & (n \le N) \\ 0 & (n > N) \end{cases}$$
$$\mu_n = \mu \qquad (n \ge 1)$$

これを式(4.3.5)と(4.3.6)に代入し, $\rho = \lambda/\mu$ を用いると, 次の結果を得る.

(4.3.10)
$$p_0 = \left\{ \sum_{i=0}^{N} \frac{N!}{(N-i)!} \rho^i \right\}^{-1}$$

(4.3.11)
$$p_n = \frac{\rho^n [N!/(N-n)!]}{\sum_{i=0}^{N} [N!/(N-i)!\rho^i]} \qquad (n = 0, 1, \ldots, N)$$

次節ではこのモデルに再び戻り, タイムシェアリング・システムの研究におけるこれの利用を論じる. なお, ここで解析した型のモデルが設定された当初の動機は, 機械が客で, 修理工がサーバである**機械保守の問題**(machine servicing problem)の研究から出ているということを述べておく.

最後の例として**複数サーバ**(multiserver)(M/M/$m$)ポアソン型待ち行列を考えよう. $m$ 個のプロセッサの存在とこれらのプロセッサが使用可能になり次第ジョブを割当てるサービス規律を仮定する. 明らかに, システム内に $n$ ($n \le m$)個のジョブしかなければ, 行列自体は空であるが, プロセッサの中の $n$ 個が稼働中である. ここでの結果を応用するため, $\lambda_n$, $\mu_n$ を次のように設定

しよう．

$$\lambda_n = \lambda \qquad (n=0,1,2,\ldots) \tag{4.3.12}$$
$$\mu_n = \begin{cases} n\mu & (1 \le n \le m) \\ m\mu & (n \ge m) \end{cases}$$

$m$ 個のプロセッサに対して利用率を新しく $\rho=\lambda/m\mu$ と定義すると，式(4.3.5)と(4.3.6)の結果から次式が得られる．

$$p_n = \begin{cases} \dfrac{(m\rho)^n}{n!}p_0 & (n<m) \\ \dfrac{m^m}{m!}\rho^n p_0 & (n \ge m) \end{cases} \tag{4.3.13}$$

ただし，

$$p_0 = \left\{\frac{(m\rho)^m}{m!(1-\rho)} + \sum_{i=0}^{m-1}\frac{(m\rho)^i}{i!}\right\}^{-1} \tag{4.3.14}$$

モーメントは，直ちに求められるとはいえ，単純な形の結果にはならない．

## 4.4 ラウンドロビン・サービス規律

ラウンドロビン(RR)・サービス規律は，タイムシェアリング・システムにおいて使用された最初のものの一つであった．それというのも，第一にその単純さのためであり，また，それがより短いジョブに対してはより短い待ち時間を与えるという特性を有しているからである．RR 規律を定義した後で，その待ち時間特性を定量化する課題に取り掛かることにする．

RR システムを図 4.4-1 に示す．そこでは，平均到着率 $\lambda$ のポアソン到着が仮定される．任意のサービス時間分布を仮定して，ジョブの順序付けが以下の

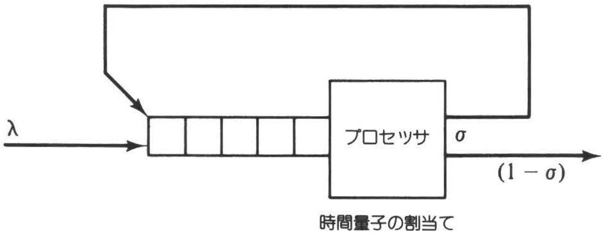



図 4.4-1 ラウンドロビン・システム

ように進められる．一つのジョブが実行のために選択されるときにはいつもそれは順序付けられた列の先頭から選び出されて**時間量子**(quantum)または**時間切片**(time slice)と呼ばれる一定量の実行時間を割当てられる． 与えられた時間量子の大きさは $Q$ 時間単位であるとする． もしジョブが時間量子の満了に先立って完了すれば， そのジョブは単にシステムから退去するだけである． もし $Q$ 時間単位後にジョブがさらに追加の実行時間を必要とするならば，行列の後尾に戻り(フィードバックされ)，次のプロセッサ割当ての順番が回って来るのを待つ．このようにして一つのジョブは，それが必要とするサービスに要する時間量子の数と同じ回数だけ繰り返して待ち行列を通過することになる．新規の到着は，到着時に行列の後尾に単に加わるだけである．

条件付待ち時間分布の基本構造を変更することなしに解析を簡単化するため，サービス時間は時間量子の大きさの整数倍であると仮定しよう．このことは， 1回分のサービスに1時間量子だけを割当てるような他の諸規律を議論する際にも引続いて仮定される． 大きさ $Q$ の時間間隔を単位とする(離散的)サービス時間の分布は，特に次の幾何分布によって定義されるものとする．

$$g_i = \sigma^{i-1}(1-\sigma) \quad (0<\sigma<1,\ i=1,\ 2,\ \ldots) \tag{4.4.1}$$

すなわち， 与えられたジョブが $i$ 時間量子(つまり， $iQ$ 時間単位)を必要とする確率は $g_i$ で与えられる． サービス時間分布の1次および2次のモーメントは次式で与えられる．

$$\mathrm{E}(S) = \sum_{i=1}^{\infty}(iQ)g_i = \frac{Q}{1-\sigma} \tag{4.4.2}$$

$$\mathrm{E}(S^2) = \sum_{i=1}^{\infty}(iQ)^2 g_i = \frac{1+\sigma}{(1-\sigma)^2}Q^2 \tag{4.4.3}$$

無記憶性のゆえに，幾何分布(あるいは，連続的サービス時間に対する指数分布)の仮定はRRシステムの解析にとって本質的である． それというのも，この特性は，一つのジョブが時間量子を――すでに受取った個数に関係なく――さらにもう1個必要とする確率は常に一定であって，その値はパラメータ値 $\sigma$ に他ならないと述べることを可能ならしめるからである．

統計的平衡状態にあるシステムへのあるランダム到着を考える．この到着ジョブは， システム内に $j$ 個のジョブが存在しているのを見出し， しかもそれ自身はサービスに $k$ 時間量子を必要とするジョブであると仮定しよう． $W_k(j)$ は，到着時にはシステム内に $j$ 個のジョブが存在するという条件を与えられた

として，この到着ジョブがシステム内で費やす時間の条件付期待値を表すものとする．そうすると，そのようなジョブのシステム内における条件付待ち時間は次式のように定義される．

$$W_k = \sum_{j=0}^{\infty} p_j W_k(j) \tag{4.4.4}$$

ここで，$\{p_j\}_{j=0}^{\infty}$ は RR システム内のジョブの個数(RR システムの長さ)の定常確率分布である†．今や，目標はパラメータ $\lambda, \sigma$, および $Q$ を用いて $W_k$ の表現式を作り出すことである．しかし，まず $W_k(j)$を算出しなければならない．実際の所，ここでの接近法は，待ち時間を求めようとしている到着ジョブに '下げ札(tag)' を付け，それをシステムを通じて追跡しようというものである．

$k$ 時間量子のサービスを必要とするジョブは待ち行列を $k$ 回通過するはずである．すなわち，そのようなジョブは待ち行列の後尾に $k-1$ 回フィードバックされる．1回の通過の長さは，そのジョブが行列に加わる瞬間からそれが次に再び行列に加わる(または，そのジョブの次の時間量子が最後のものであるならば，退去する）瞬間までの時間である．この定義に従えば，各通過は，最初の通過を除いて，整数個の時間量子を要する††．

$U_i(j)\ (i = 1, 2, \ldots, k)$は，到着時にシステム内に $j$ 個のジョブがあると仮定して，$i$ 回目の通過に要する時間を表す確率変数であるとする．明らかに次式が成り立つ．

$$W_k(j) = \sum_{i=1}^{k} \mathrm{E}[U_i(j)] \tag{4.4.5}$$

ところで，もし $U_i(j) = x (i \geq 2)$ならば，$(x/Q) - 1$ は $i$ 回目の通過に際し下げ札付きのジョブの前にあるジョブの数を表す．$(i+1)$回目の通過において1時間量子を受け取る前に，これらのうち平均すると $\sigma[(x/Q) - 1]$ 個が戻っているはずである(ここが幾何分布の無記憶性を発動する所である)．また，$i$ 回目の通過中に平均すると $\lambda x$ 個の到着が生じるから，$(i+1)$回目の通過に対しては次式が成り立つ．

$$\mathrm{E}[U_{i+1}(j) \mid U_i(j) = x] = \sigma Q\left(\frac{x}{Q} - 1\right) + \lambda x Q + Q$$

---

† 198頁の脚注を見よ．

†† **訳注** 最初の通過では，到着時点が他のジョブの実行途中であるならば，その時間量子の残り時間は $Q$ より小であるからである．

この式から次式を得る.

(4.4.6)　$E[U_{i+1}(j)] = (\lambda Q + \sigma)E[U_i(j)] + Q(1-\sigma) \quad (i = 2, 3, \ldots)$

最初の通過では $j$ 個のジョブが下げ札付きのジョブの前にあるという条件が与えられているから，次式が成り立つ.

(4.4.7)
$$E[U_2(j)] = \lambda Q E[U_1(j)] + Q(\sigma j + 1)$$

簡単な帰納法により，$E[U_i(j)]\,(3 \le i \le k)$ に対して次のように陽に表現された式を得ることができる.

(4.4.8)
$$E[U_i(j)] = (\lambda Q + \sigma)^{i-2} E[U_2(j)] + Q(1-\sigma)\frac{1-(\lambda Q+\sigma)^{i-2}}{1-\lambda Q-\sigma}$$

式(4.4.5)に代入し加算を実行すると，次式が求められる.

(4.4.9)
$$W_k(j) = E[U_1(j)] + \frac{(k-1)Q}{1-\rho} + Q\left[\lambda E[U_1(j)] + \sigma j - \frac{\rho}{1-\rho}\right]\frac{1-\alpha^{k-1}}{1-\alpha} \quad (k = 1, 2, \ldots)$$

ただし，

(4.4.10)
$$\alpha = \lambda Q + \sigma$$

また，$\rho$ はこのシステムの利用率である．すなわち，

(4.4.11)
$$\rho = \frac{\lambda Q}{1-\sigma}$$

これを式(4.4.4)に代入し，次式

$$W_1 = \sum_{j=0}^{\infty} p_j E[U_1(j)]$$

に注意すると，次の結果を得る.

(4.4.12)
$$W_k = W_1 + \frac{(k-1)Q}{1-\rho} + Q\left[\lambda W_1 + \sigma \bar{n} - \frac{\rho}{1-\rho}\right]\frac{1-\alpha^{k-1}}{1-\alpha} \quad (k \ge 2)$$

ここで，$\overline{n}$ は定常分布$\{p_j\}$の平均である．ところがこの分布は M/M/1 型システムに適用されるものと正確に同じである．このことは，たとえば，両システムにおいて全稼働期間内の退去間隔がパラメータ $\sigma$ の幾何分布を持つという事実からも納得することができる．これゆえに，式(4.2.21)における $\mathrm{E}(S^2)$ に対して式(4.4.3)を用いると，次式を得る．

(4.4.13)
$$\overline{n} = \rho + \frac{(1+\sigma)\rho^2}{2(1-\rho)}$$

さらに，1回目の通過時間の期待値については次式が成立する．

(4.4.14)
$$W_1 = \mathrm{E}(Q_r) + \overline{n}_q Q + Q$$

ここで，$\mathrm{E}(Q_r)$は到着時に(もし存在するならば)進行中の時間量子が終了するまでの平均時間である．時間量子は固定長であるので，式(4.2.28)から $\mathrm{E}(Q_r) = \rho Q/2$ である．$\overline{n}_q = \overline{n} - \rho$ を使うと次式を得る．

(4.4.15)
$$W_1 = \frac{1-\rho}{2}Q + \overline{n}Q$$

式(4.4.13)および(4.4.15)を用いれば，$W_k$ は，式(4.4.12)によって完全に決定される．

諸パラメータ値を固定するならば，$W_k$ の $k$ への依存は，$k$ が大きいときには1次の項$(k-1)Q/(1-\rho)$によって支配されることが式(4.4.12)から容易にみてとれる．さらに，平均サービス時間 $Q/(1-\sigma)$を一定値に固定したまま，$Q$ を零に近付けたときの式(4.4.12)の極限を考えるならば，この1次依存は厳密なものであるといえることを示そう．

まず，$Q/(1-\sigma)$を固定したまま $Q$ を微小にすると，ジョブは一巡する毎に微小量のサービスを受けながら数多の回数にわたって循環するということに注意しよう．明らかに，行列内のジョブの相対的な位置は，$Q$ を小さくすると段々影響を持たなくなる．こうして，極限 $Q \to 0$ においては幾何分布するサービス時間は同じ平均を持つ指数分布するサービス時間になってしまい，その結果，システムは，第3章の**プロセッサ共有方式**(processor sharing)を動的スケジューリング規律の下で実現したものに相当することになる．さらにはっきり述べれば，もしある時点でプロセッサ共有(PS)システム内に $n$ 個のジョブが存在するならば，各々のジョブは，プロセッサ独占状態にあるジョブが受けるサービスの $1/n$ の率でサービスを受けつつあるということになる．

式(4.4.12)から対応する条件付待ち時間の表現式を求めるため，サービス必要量を $t = kQ$ で，またこれに伴う平均待ち時間を $W(t)$ で表そう．非形式的に述べると，$t/Q$ を式(4.4.12)の $k$ に代入し，$\rho = \lambda Q/(1-\sigma)$ を一定に保ったまま $Q$ を零に近付けて極限を取る．式(4.4.12)の第2項が極限においても消滅しない唯一の項であることを確かめるのは困難なことではない．その結果，次式が求められる．

$$W(t) = \frac{t}{1-\rho} \tag{4.4.16}$$

よって，指数分布に従うという仮定の下で PS システム内で費やされる平均時間は，必要とするサービス量に直接比例する．$W(t)$ はサービス時間の分散に影響されないということに特に注意しよう．直観的には，これは，FIFO システムにおいて見られたより長いジョブがより短いジョブを妨害するという現象を防ぐ共有方式の特性から来ている．PS および FIFO システムにおける平均の長さ($\overline{n}$)は指数分布する到着時間間隔およびサービス時間の仮定の下では同じである．したがって，リトルの公式を適用することによって両システムに対する平均(条件付きでない)待ち時間も同一になり，次式のように与えられることが分る．

$$W = \frac{\mathrm{E}(S)}{1-\rho} \tag{4.4.17}$$

ここで，$\mathrm{E}(S)$ はサービス時間分布の平均であり，$\rho = \lambda\mathrm{E}(S)$ は利用率である．PS システム内で費やされる平均時間(平均滞留時間)は，いかなるサービス時間分布(ただし，平均値は $\mathrm{E}(S)$ とする)に対しても式(4.4.17)によって与えられることが文献[10]で示されている．

前節で述べたように，比較的に小数の利用者端末を持つようなタイムシェアリング・システムに対するモデルでは有限入力源を仮定するのがより現実的である．プロセッサ共有規律の下での有限入力源システムを4.3節のモデルを用いて研究することができる．なぜかというと，式(4.3.10)および(4.3.11)における結果は，プロセッサ共有規律が全稼働期間内での退去時間間隔の分布(指数分布)を変えてしまう訳ではないので，このような目的に対しても依然有効であるからである．有限入力源タイムシェアリング・モデルがこれらのシステムを代表しているという見方から，システム内における平均滞留時間はしばしば**平均応答時間**(mean response time)と呼ばれ，一方，前の要求(request)に

対するサービスが完了した時点から新しい要求が生成されるまでの時間は**思考時間**(think time)と呼ばれる．

平均応答時間 $W$ は次の論法を用いて容易に算出される．平均思考時間は各端末に対して $1/\lambda$ であるから，ある利用者のシステム内滞留時間の(平均の)割合は $W/(W+1/\lambda)$ で与えられる．それゆえ，システムによって処理中の利用者の平均数 $\overline{n}$ は次のように書くことができる．

$$\overline{n}=N\frac{W}{W+1/\lambda} \tag{4.4.18}$$

式(4.4.18)を式(4.3.11)の分布から計算される平均に等しいと置くと次のようになる．

$$N\frac{W}{W+1/\lambda}=\sum_{n=0}^{N}np_n=p_0\sum_{n=1}^{N}\frac{N!}{(N-n)!}n\rho^n$$

ただし，

$$p_0=\left\{\sum_{n=0}^{N}\frac{N!}{(N-n)!}\rho^n\right\}^{-1} \tag{4.4.19}$$

$W$ について解けば，決まりきった代数演算を多少行った後に次の結果が得られる．

$$W=\frac{N(1/\mu)}{1-p_0}-\frac{1}{\lambda} \tag{4.4.20}$$

これまでのタイムシェアリング解析の主要な短所は，プロセッサから一つのジョブを取り除いて次に実行するべきジョブを搭載(load)する切換え機構によって一般的に生じる費用(時間遅れ)を考慮に入れなかったという欠陥があったことである．そのような費用は，実行中の全てのプログラムを記憶するかあるいは少なくとも入/出力と実行の高多重度の並行化(overlapping)を保証するのに十分な主記憶が存在するようなシステムにおいてのみ無視できるであろう．この時間遅れを包含するための一つの平凡ではあるが近似的な技法は，このいわゆるプログラム**交換**(スワッピング，swapping)による時間遅れに当たる一定の初期時間間隔を時間量子が常に含むと仮定することである．この技法に近似性を齎す一つの原因は，システム内に唯一つのジョブが存在している間はこのジョブが実行の最中に主記憶から外へ転送されたりまた主記憶内へ転送され

たりすると仮定することは本来できないという事実にある．明らかに，この近似の精度は，システムに対する($\rho$によって測られるような)需要が増加するにつれて高くなる．交換の費用を考慮に入れるこの方法を用いると，数学的モデルに対してここで計算した結果と観測値とが非常によく一致することが汎用タイムシェアリング・システムについて行われた実験で示されている[11]．

近似性のもう一つの原因が，交換時間を一定値に固定するという仮定に潜んでいる．しかしながら，一定値の交換時間という仮定は，プロセスの交換に要する時間の支配的な部分を**準一定値**(near-constant)である探索時間が占めるようなディスク装置，または同様な回転待ち時間を持つドラム装置を補助記憶とするようなシステムに対しては使用に堪える近似を齎すはずである†．このようなシステムに関するさらに詳細な事柄については第5章で議論することにしよう．

上述した二つの原因に基づく近似性を排除して交換時間の精密なモデル化を行うと，必然的にマルコフ性を持たない待合わせ過程に行き着いてしまう．このような一般モデルへの接近法はいずれも，4.2節で行われた解析と同様に，マルコフ連鎖の理論の応用から成っている．一つのマルコフ連鎖は，併発する交換に要する時間をも含んで，各時間量子内でサービスを受けた直後の時点で定義することによって得られる．有限入力源モデルの場合には有限マルコフ連鎖となることを注意しておく．したがって，定常確率分布は，1段遷移確率で定式化される行列(ただし，大次元行列となる可能性がある)の逆行列を取ることにより常に求めることができる．しかし，一般には，このようなマルコフ連鎖による有限入力源モデルへの接近法は直接には解釈し難いような厄介な待ち時間についての結果を導出しがちである．そうはいうものの，他方では，効率的な計算方法も開発されつつある．当然のことながら，数値的研究はシステムの振舞のさらに現実的な記述を得ることを指向している．

無限入力源RRシステムに対する条件付待ち時間(の期待値)は，離散時間モデル(すなわち，到着時間間隔とサービス時間はいずれも幾何分布に従う)を用いることによりKleinrock[12]が初めて求めたのであるが，本節での解析は，連続時間におけるRRモデルに対して文献[13, 14]が用いた取扱い方により近い扱い方をしている．プロセッサ共有方式に対する結果は，RR方式に対する上記の結果の極限として得られた[13, 14, 15]のであるが，RRシステム

---

† **訳注**　特に回転待ち時間については，少数回，たとえば1回の回転待ち時間を準一定値とはとても考えられない．しかし，1個のプログラムの搭載にある程度多数回の読込み操作が必要である場合を想定するならば，大数の法則によって準一定値と見なせよう．

[16]およびプロセッサ共有システム[17]に対して条件付待ち時間の完全な分布(実際は，そのラプラス変換)が決定されたということは注目に値する．(交換時間に関するパラメータを含む)有限入力源 RR モデルに対する条件付待ち時間は，Coffman[18]，および Krishnamoorthi と Wood[19]によって最初に求められた．上記の諸論文の出現以後，数多くの文献が RR システムとその変形を取扱っている．本節で言及した文献以外の参考文献に対しても興味を持つ読者は McKinney[20]による概観を参照するとよいであろう．

## 4.5 横取り割込み不可の優先権待ち行列

本節では，ジョブに付与されている優先権に基づいて優遇処置を取るシステムを考察することにする．ジョブの優先権は，到着時に定められる整数で表示されると仮定する．また，今後 $k$ 番目の優先権クラスというときは，それは優先権 $k(k=1, 2, \ldots)$を持つジョブのクラスを指すことにする．多くの文献で行われている習慣にならって，あるジョブが他のあるジョブよりも高い優先権を持つのは，前者がより小さい整数で表示される優先権クラスに属するときであると定めよう．

研究の対象となる優先権待ち行列システムは**図 4.5-1** に示されている．図では，異なった高さに位置するように描かれた待ち行列がそれぞれ異なった優先権クラスに対応している†．サービス規律は，あるジョブが完了したときには，必ず次に，最高優先権の（すなわち，空でなく，しかも最も下位に位置する）

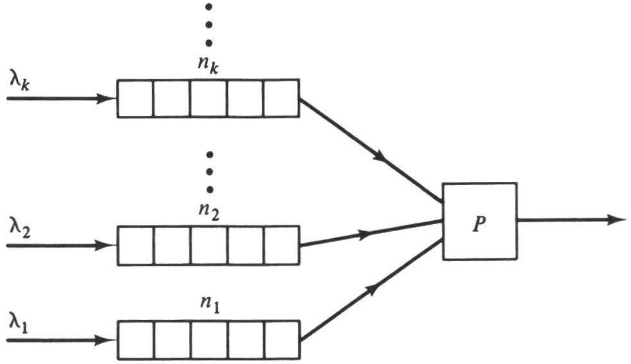



**図 4.5-1** 横取り割込み不可の優先権待ち行列

† **訳注** 最下位に描かれた待ち行列が優先権 1 のクラスに対応し，下から $k$ 番目の位置の待ち行列は優先権 $k$ のクラスに対応する．優先権 $k$ のクラスに対応する待ち行列をレベル $k$ の待ち行列ともいう．

待ち行列の先頭にあるジョブにプロセッサを割当てようとするように働くと仮定する．一度あるジョブの処理がプロセッサ上で開始されたならば，その処理は完了するに到るまで走ることが許されている．つまり，サービス規律は横取り割込み不可である．異なる優先権クラスの各々に対して独立なポアソン到着を仮定する．$k(k=1, 2, \ldots)$番目のクラスの到着率は $\lambda_k$ であるとする．各優先権クラスには任意の互いに異なり得るサービス時間分布が対応しているものとする．$k$ 番目のクラスに対する分布の $i$ 次のモーメントは $\mathrm{E}(S^i{}_k)$で表すことにする．[なお，いつものように $\mathrm{E}(S_k)=\mathrm{E}(S^1{}_k)$と置く．]

さて，統計的平衡状態における $k$ 番目の待ち行列へのランダム到着の平均待ち時間 $W_k$ を考えよう．サービス規律の定義からまず次の式が成り立つ．

$$W_k = V_k + \mathrm{E}(S_k) \tag{4.5.1}$$

ただし，$V_k$ は優先権 $k$ のジョブの行列における平均待ち時間である．そこで，次に $V_k$ の表現式を見出すことにする．

$n_i(i=1, 2, \ldots)$は，ある優先権 $k$ のジョブ(前と同様にこれを下げ札付きのジョブと呼ぶ)が到着時にレベル $i$ の待ち行列内に見出すジョブの個数であるとする．下げ札付きのジョブの処理は，レベル1から $k$ までの待ち行列内にあるジョブ，すなわち，同等以上の優先権を持つジョブの全てが完了してしまうまでは開始できないことが明らかである．それだけに止まらず，下げ札付きのジョブは，その待ち時間内に到着する優先権1から $k-1$ までのジョブの全ての完了をも待たなければならない．$V_k$ の計算のため，下げ札付きのジョブの到着時にはシステム内に既に存在していて先に実行されなければならない優先権1から $k$ までのジョブが全部で $n_1+n_2+\cdots+n_k$ 個あるとし，しかもこれらをまとめて最初に実行してしまうと仮定しよう．これに要する時間間隔の期待値を $V'_k$ とする．これに続いて，下げ札付きのジョブが行列内にある間に到着する優先権より高いジョブの全てをプロセッサが実行するのに要する時間間隔の期待値を $V''_k$ で表す．$V_k$ を次式のように三つの要素に分けて書くことができる．

$$V_k = \mathrm{E}(S_r) + V'_k + V''_k \tag{4.5.2}$$

ただし，$\mathrm{E}(S_r)$は，到着時刻にプロセッサ上にあったジョブの実行時間の残余の平均であり，式(4.2.29)により次式のように与えられる．

$$\mathrm{E}(S_r) = \frac{1}{2}\sum_{k=1}^{\infty}\lambda_k \mathrm{E}(S_k{}^2) \tag{4.5.3}$$

式(4.5.2)は，下げ札付きのジョブが完了を待たなければならない一連の処理時間の並べ換えに基づいて立てられているけれども，そうすることの妥当性は，到着過程がポアソン的であって，サービス機構とシステムの状態には独立であるという事実に存する．定義に従えば，

$$V'_k = \sum_{i=1}^{k} \mathrm{E}(n_i)\mathrm{E}(S_i) \tag{4.5.4}$$

が成立する．$\rho_i = \lambda_i \mathrm{E}(S_i)$と置き，リトルの公式を最初の $k$ レベルの待ち行列に適用すると，次式が得られる．

$$V'_k = \sum_{i=1}^{k} \lambda_i V_i \mathrm{E}(S_i) = \sum_{i=1}^{k} \rho_i V_i \tag{4.5.5}$$

一方，下げ札付きのジョブが行列内に滞留している間に到着する優先権 $i$ のジョブの平均個数は $\lambda_i V_k$ であるから，

$$V''_k = \sum_{i=1}^{k-1} \lambda_i V_k \mathrm{E}(S_i) = \sum_{i=1}^{k-1} \rho_i V_k \tag{4.5.6}$$

を得る．式(4.5.5)と(4.5.6)を式(4.5.2)に代入すると，次式が得られる．

$$V_k = \mathrm{E}(S_r) + \sum_{i=1}^{k} \rho_i V_i + \sum_{i=1}^{k-1} \rho_i V_k$$

すなわち，

$$V_k = \frac{\mathrm{E}(S_r) + \sum_{i=1}^{k} \rho_i V_i}{1 - \sum_{i=1}^{k-1} \rho_i} \tag{4.5.7}$$

$\mathrm{E}(S_r)$に対して式(4.5.3)を用いると，式(4.5.7)の解は次式で与えられることが帰納法により容易に示される．

$$V_k = \frac{\sum_{i=1}^{\infty} \lambda_i \mathrm{E}(S_i^2)}{2(1-\beta_k)(1-\beta_{k-1})} \tag{4.5.8}$$

ただし

$$\beta_j = \sum_{i=1}^{j} \rho_i \tag{4.5.9}$$

以上の結果を最初に導出したのは Cobham [21] である．なお，横取り割込み不可の優先権待ち行列に対する待ち時間分布のラプラス変換が Kesten と Runnenberg [22] によって最初に得られている．

横取り割込み不可の優先権待ち行列に対する上述の結果を利用すると，いわゆる**最短処理時間**(shortest-processing-time, SPT) 規律に対する類似の結果がかなり簡単に得られる．この規律は，ジョブの実行時間を到着の時点で予知できると仮定している．この SPT 規律の下では，プロセッサがある一つのジョブの実行を完了すると，必ず，待機中のジョブ(存在するならば)の中で最小の実行時間を持つものの一つが次に実行されるジョブとなる．

平均待ち時間を求めるために，ひとまず，離散的サービス時間を仮定して問題を単純化しよう．ジョブが実行に $iQ$ 単位の時間を要する確率は $g_i (i = 1, 2, \ldots)$ であるとして，このサービス時間分布の $i$ 次のモーメントを前と同様に $\mathrm{E}(S^i)\ (i = 1, 2, \ldots)$ で表す．システムへのジョブの総到着率を $\lambda$ で表すと，実行に $k$ 時間量子を必要とするジョブの到着率は $\lambda g_k$ である．定常確率分布に従ってポアソン過程をいくつかの支流過程に分解すると，互いに独立なポアソン過程が生じる(4.1.2 節参照)から，図 4.5-1 において $\lambda_k = \lambda g_k$ と置けば，SPT システムが表現されることになる．$\lambda g_k$ と $iQ\lambda g_i$ をそれぞれ $\lambda_k$ と $\rho_i$ に代入すると，式(4.5.8)から，$k$ 時間量子を要するジョブの行列内における平均待ち時間が次の通り得られる．

$$V_k = \frac{\lambda \mathrm{E}(S^2)}{2(1 - \lambda H_k(S))(1 - \lambda H_{k-1}(S))} \tag{4.5.10}$$

ただし，

$$H_j(S) = \sum_{i=1}^{j} (iQ) g_i \tag{4.5.11}$$

連続的サービス時間分布 $B(x)$ に対する結果は，非形式的に述べるならば，次のようにして求められる．(指数分布を離散化したものが幾何分布であるのと同じ意味で) $g_i$ は $B(x)$ を離散化して得た確率塊関数であると考え，この条件の下で時間量子の大きさ $Q$ を微小にすることによって求める．前節のプロセッサ共有の場合と同様に，式(4.5.10)の極限 $(Q \to 0)$ を取ると，実行時間が $t$ であるジョブの行列内における条件付待ち時間(Phipps [23] の結果)が次の通り得られる．

$$V(t) = \frac{\lambda \mathrm{E}(S)}{2\left[1 - \lambda \int_0^t x dB(x)\right]^2} \tag{4.5.12}$$

## 4.6　最短経過時間規律

本節では，タイムシェアリング・アルゴリズムの研究を，ジョブの走行時間はあらかじめ知ることができないという仮定の下で続けることにしよう．4.4 節で述べた RR 規律と並んで，実行時間の短いジョブを厚遇するもう一つの手段であるいわゆる**最短経過時間**(shortest-elapsed-time, SET)規律の離散版が研究されている．

**図 4.6-1** には離散的 SET 規律を用いるシステムが示されている．図 4.6-1 の構造と下記の定義から，離散的 SET 規律は，**多レベル・フィードバック型待ち行列規律**(multiple-level, feedback queueing discipline)とも呼ばれている．特に本節で解析を行うために選んだシステムでは，ジョブは，速度(到着率)$\lambda$ のポアソン流れを形成して到着し，離散的ではあるが他の点では一般の確率分布に従うサービス時間を伴うということが仮定されている．前と同じく

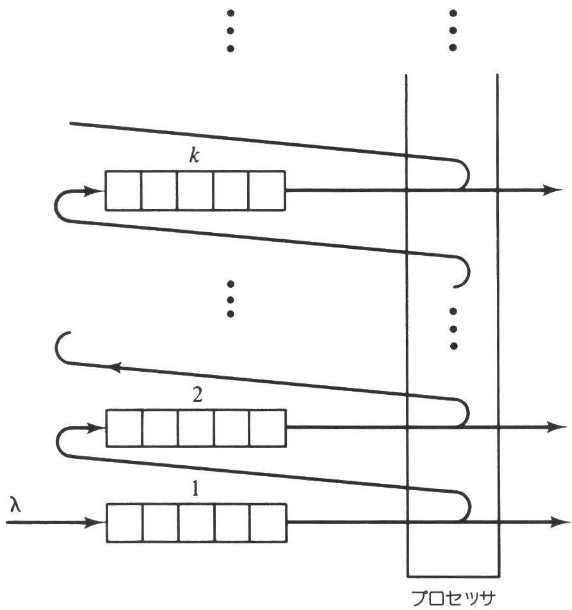



**図 4.6-1**　離散的 SET システム

$g_i (i = 1, 2, \ldots)$はジョブが実行に $iQ$ 単位の時間を要する確率を表し，$G(k) = \sum_{i=1}^{k} g_i$ はその累積分布関数を表す．

サービス規律の一般的な記述は以下の通りである．プロセッサがある与えられたジョブに割当てられた時間量子を終了したとき，現在待機中の全てのジョブの中で最小数の時間量子しか受取っていなかったものが次の1時間量子を割当てるジョブとして選ばれる．もしも互角のジョブがいくつか存在するならば，その中で最も早期に到着したものが選ばれる．この規律は，既に $k-1$ 個の時間量子を受取りさらに $k$ 個目の時間量子を受取るために待機中のジョブの集合$(k = 1, 2, \ldots)$を，単に，レベル $k$ の待ち行列(優先権 $k$ のクラス)に対応させることによって，図4.6-1に示された優先権待ち行列に帰着させて考察することができる．レベル $k$ にあるジョブは，レベル1からレベル $k-1$ までの待ち行列が全て空でなければ新たに時間量子を割当てられて実行を再開することができない．ジョブが割当てられた $k$ 個目の時間量子を終了したとき，そのジョブは，全ての処理を完了したためにシステムから退去するか，あるいはさらに $k+1$ 個目の時間量子の割当てを受けるためにレベル $k+1$ の待ち行列の後尾に付くかのいずれかである．

RR規律とSET規律の大まかな比較によって，前者よりも後者の方が，長時間ジョブにはさらに長時間待つという犠牲を払わせるであろうけれども，短時間ジョブに対してはもっと短時間の待ちで実行を完了せしめるという優遇措置を取っているであろうことがあらかじめ知られる．そこで，待ち時間の比較を明確に行えるようにするため，以下ではSET規律に対する結果を求めることにする．実行に $k$ 時間量子を必要とするランダム到着の平均待ち時間について，まず，次式のように書くことができる．

$$W_k = V_k + kQ \tag{4.6.1}$$

$V_k$ を計算するために，これを次のように分解する．

$$V_k = V'_k + V''_k \tag{4.6.2}$$

ただし，$V'_k$ と $V''_k$ を以下のように定義する．$V'_k$ は，到着時に進行中であった時間量子をまず完了させ，さらに到着時に既にシステム内にあったジョブの $k$ 時間量子までの実行を完了するのに要する平均の時間である．すなわち，$V'_k$ は，進行の途中にあったジョブをひとまず終らせるまでに要する時間の平均と到着時にレベル1から $k$ までの待ち行列中にあった全てのジョブにサービスを施すのに要する時間の平均の和である．

次に，本節で利用する優先権規律によれば，新到着者(下げ札付きのジョブ)が行列内で待ちつつあるかまたは最初の $k-1$ 時間量子を受取りつつある間に到着するジョブは全て，（下げ札付きのジョブが最後($k$ 個目)の時間量子を割当てられるまでに）最大 $k-1$ 時間量子までの実行時間を割当てられてしまわなければならない．すなわち，そのようなジョブが$(k-1)Q$ 単位を超える実行時間を必要とするときは，レベル $k$ の待ち行列へ昇り，下げ札付きのジョブの背後に付かなければならない．このようなその後の到着者によって必要とされる上述の実行時間の総量の平均値を $V''_k$ と定義する．

ところで，どの1個のジョブにも $mQ$ 単位(供給限度)までのプロセッサ時間を割当てると仮定しよう．この条件の下で1個のジョブが実行のために消費するプロセッサ時間の平均値を $\mathrm{E}_m(S)$ で表すと，$\mathrm{E}_m(S)$について次式が成り立つ．

$$\mathrm{E}_m(S) = \sum_{i=1}^{m} (iQ)g_i + mQ[1 - G(m)] \tag{4.6.3}$$

後の使用のために2次のモーメントを書いておくと，

$$\mathrm{E}_m(S^2) = \sum_{i=1}^{m} (iQ)^2 g_i + (mQ)^2[1 - G(m)]$$

これを用いると，$V''_k$ の定義から直ちに次式が得られる．

$$V''_k = \lambda[V_k + (k-1)Q]\mathrm{E}_{k-1}(S) \tag{4.6.4}$$

$V'_k$ を計算する問題を簡単にするため，システムを次のように修正しよう．すなわち，第1の待ち行列レベルでサービスされるジョブには最大 $kQ$ 単位まで実行時間を割当て，もっと高レベルでサービスされるものには，1時間量子のサービスを割当てるような SET システムの修正版を考える†．この新システムを修正 SET システムと呼ぶことにする(図4.6-2参照)．実行に $k$ 時間量子を必要とするジョブにとって，このジョブが新システムのレベル1の待ち行列内で消費する時間の平均は正に $V'_k$，すなわち，旧システムにおいて到着時に存在したジョブ(実行途中のジョブとレベル1から $k$ までの待ち行列中にあったジョブ)が最大限 $k$ 時間量子を受取ってしまうのを待つのに要する時間の平均に等しいことに注目しよう．さて，修正 SET システムでは次の式が成立

† **訳注** この修正版におけるレベル $m (m \geq 2)$ の待ち行列は元のシステムのレベル $k+m-1$ の待ち行列に対応する．

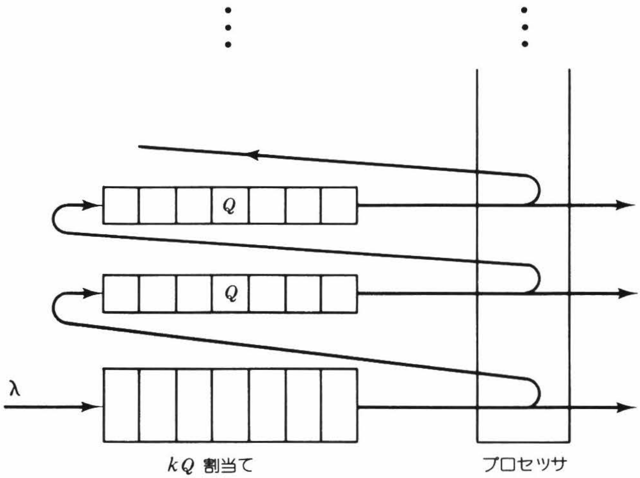



図4.6-2　修正 SET システム

する．

$$V'_k = \mathrm{E}'(S_r) + N'_1\mathrm{E}_k(S) \tag{4.6.5}$$

ただし，$\mathrm{E}'(S_r)$は到着時に進行の途上にあったサービスをまず完了するのに要する時間の平均であり，$N'_1$は到着時に新システムにおけるレベル1の待ち行列中にあったジョブの平均数である．リトルの公式を使うと次式のようになる．

$$V'_k = \mathrm{E}'(S_r) + \lambda V'_k\mathrm{E}_k(S) \tag{4.6.6}$$

すなわち，

$$V'_k = \frac{\mathrm{E}'(S_r)}{1 - \lambda\mathrm{E}_k(S)} \tag{4.6.7}$$

$\mathrm{E}'(S_r)$の計算は，修正 SET システムでは実行中のサービスには二つのクラスがあることを考慮して行わなければならない．すなわち一つは，レベル1の待ち行列での待機が終了し，$kQ$ 単位の時間を割当てられて実行の途上にあるジョブに対するサービスであり，他の一つは，高レベルの待ち行列での待機が終わり，1時間量子の割当てを受けて実行の途上にあるジョブに対するサービスである．新システムにおけるレベル1の待ち行列への到着率は単に $\lambda$ である

が，レベル $i(i=2, 3, \ldots)$ の待ち行列への到着率 $\lambda'_i$ は $(k+i-1)Q$ 単位以上の実行時間を必要とする到着ジョブの割合によって定まる．すなわち，

(4.6.8) $$\lambda'_i = \lambda \sum_{j=k+i-1}^{\infty} g_j = \lambda[1-G(k+i-2)] \quad (i=2, 3, \ldots)$$

ところで，レベル1の待ち行列に対する到着過程はポアソン的であるけれども，もっと高レベルの待ち行列への到着はポアソン過程を形成しない．(高レベルの待ち行列への到着はプロセッサの全稼働期間内においてのみ生じることに注意しよう.)それにもかかわらず，下げ札付きのジョブはランダムな(すなわち，ポアソン的な)到着であるという仮定によって，式(4.2.29)に到る議論は依然として有効である†．したがって，次式が成立することになる．

(4.6.9) $$\mathrm{E}'(S_r) = \frac{\lambda}{2}\left[\mathrm{E}_k(S^2) + Q^2 \sum_{i=0}^{\infty} [1-G(k+i)]\right]$$

式(4.6.7)と(4.6.4)を式(4.6.2)に代入すると次式を得る．

(4.6.10) $$V_k = \lambda \mathrm{E}_{k-1}(S) V_k + \frac{\mathrm{E}'(S_r)}{1-\lambda \mathrm{E}_k(S)} + \lambda \mathrm{E}_{k-1}(S)(k-1)Q$$

$V_k$ についてこの式を解き，さらに式(4.6.9)を用いると，$V_k$ に関する最終結果が得られる．

(4.6.11)
$$V_k = \frac{\lambda\left[\mathrm{E}_k(S^2) + Q^2 \sum_{i=k}^{\infty} [1-G(i)]\right]}{2[1-\lambda \mathrm{E}_{k-1}(S)][1-\lambda \mathrm{E}_k(S)]} + \frac{\lambda \mathrm{E}_{k-1}(S)}{1-\lambda \mathrm{E}_{k-1}(S)}(k-1)Q$$

式(4.6.1)を用いると，$W_k$ に対する結果が得られる．

(4.6.12)
$$W_k = \frac{\lambda\left[\mathrm{E}_k(S^2) + Q^2 \sum_{i=k}^{\infty} [1-G(i)]\right]}{2[1-\lambda \mathrm{E}_{k-1}(S)][1-\lambda \mathrm{E}_k(S)]} + \frac{(k-1)Q}{1-\lambda \mathrm{E}_{k-1}(S)} + Q$$

サービス時間分布が連続的であると仮定される基本 SET 規律†† に対する結果は，サービス時間分布を $B(x)$ と置き，$t=kQ$ を固定したまま極限 $Q \to 0$

---

† 最短残存処理時間規律について考察する次節においても同様の見方をする，この見方の厳密な証明は文献[8]に見られる，

†† **訳注** 離散版でない本来の連続版 SET 規律を基本 SET 規律と呼んでいる．

を取ることにより式(4.6.12)から容易に得られる．特に，$t$ 時間単位のサービスを必要とするジョブのシステム内における待ち時間の平均は次式で与えられる†．

$$(4.6.13)\qquad W(t)=\frac{(\lambda/2)\left[\int_0^t x^2dB(x)+t^2[1-B(t)]\right]}{\left[1-\lambda\int_0^t xdB(x)-\lambda t[1-B(t)]\right]^2}+\frac{t}{1-\lambda\int_0^t xdB(x)-\lambda t[1-B(t)]}$$

式(4.6.11)と(4.6.12)の結果は，レベル毎に任意の大きさの時間量子を仮定したときの待ち時間分布のラプラス変換と一緒に Schrage [24] によって得られた．

前述したように，SET 規律によれば，常に，最小の実行時間しか受取っていないジョブにプロセッサが割当てられる．さらに，SET 規律に従えば，互角のジョブが存在する場合には，最小の実行時間しか受取っていないそのようなジョブの集合が 4.4 節で議論したプロセッサ共有モードでプロセッサを共用しなければならない．また，新しい到着は，常に，そのときまでプロセッサを共有していたジョブの集合を排除してプロセッサを独占する．この新到着者は下記の三つの事象のいずれかが発生するまでプロセッサを占有し続ける．すなわち，1)自身が退去するか，2)次の到着者が出現するか，あるいは，3)到着時に割込んで排除したジョブのそれぞれが既に受取った量と等しい量のサービスを取得する．3)の事象が発生した後は，到着時に排除したジョブの各々と共にプロセッサを分ち合わなくてはならない．SET 規律の下では，プロセッサ共有が生じる結果，行列における待ち時間はプロセッサ共有のないシステムにおけるそれと同じにはならないことは明らかである．

## 4.7　最短残存処理時間規律

処理時間があらかじめ知らされていて，しかも横取り割込み可の規律が採用可能であるとき，SPT 規則には考慮すべき基本的な横取り割込み可能版が二

---

† **訳注**　原文では式(4.6.13)の右辺第1項の分子は 単に

$$(\lambda/2)\int_0^\infty x^2dB(x)$$

となっている．これは明らかに誤りであるので本文のように訂正する．

つある．この二者の中で，いわば最も直接的であり単純にジョブを順序付けるのは，必要な場合には横取り割込みを許すことによって，どの時点を取っても，システム内に存在しているジョブの中で**もともとの**サービス要求量が最小であったものが実行中であるようにする版である．一方，定常状態におけるシステム内の平均個数を減少させることが関心の的であるならば，SPT 規則の第二の横取り割込み可能版，すなわち，**最短残存処理時間**(shortest-remaining-processing-time, SRPT)規則を採用することができる．SRPT 規律によりジョブは次のように順序付けられる．すなわち，いかなる時点を取っても，システム内に現在存在している全てのジョブの中で最小の残存処理時間を持つものが必ずプロセッサ上のジョブになるように順序付けが行われる．SRPT による順序付けの結果，システム内個数の平均は，事実，あり得る全てのサービス規律の中では最小になることが知られている[25]．この性質は重要であるので，本節の残りを SRPT 規律の解析に費やすことにする．

以前と同様に，ジョブの実行には整数個の時間量子から成るサービス時間が必要であると仮定して，まず，離散版の SRPT 規律を考察することにする．しかし，今回は，時間量子の途中において横取り割込みが可能である，すなわち，プロセッサ上のジョブが残しているサービス量よりも少量のサービスしか必要としないジョブが新たに到着したならば，後者は前者をプロセッサ上から直ちに排除して横取り割込みを行うと仮定すると問題が多少簡単化される．また，以前と同様に，ジョブの実行に要する時間量子の個数の一般確率塊関数を $g_i$ で表し，$G(k) = \sum_{i=1}^{k} g_i$ はその累積分布関数であるとする．

離散的 SRPT 規律の大要は図 4.7-1 に示されている．そこでは，ポアソン到着機構は，待ち行列レベル $1, 2, \ldots, k, \ldots$ のそれぞれに対する別々のポアソン到着に分解されていて，レベル $k$ の待ち行列に対する到着率は $\lambda_k$ で表されている．到着ジョブが参入する待ち行列のレベルは，そのジョブが必要とするサービスの量によって決定され，$k$ 時間量子を必要とするジョブはレベル $k$ の待ち行列に配置される．このことから，$\lambda_k = \lambda g_k (k = 1, 2, \ldots)$ が成り立つ．レベル $k$ の待ち行列のサービス受領位置(the service point of the $k$th queue level)† に位置するジョブは，レベル 1 からレベル $k-1$ までの待ち行列が空になるまでは，実行を始めることができない．レベル $k$ で与えられた時間量子を完了すると，直ちにジョブはレベル $k-1$ のサービス受領位置へ移動し，引き続いてこのジョブに対する次の時間量子が開始される．前述のように，よ

† **訳注** サービス受領位置は，対応している待ち行列の外部にある位置でなく，待ち行列の先頭の位置であると考えることにして訳文を構成してある．

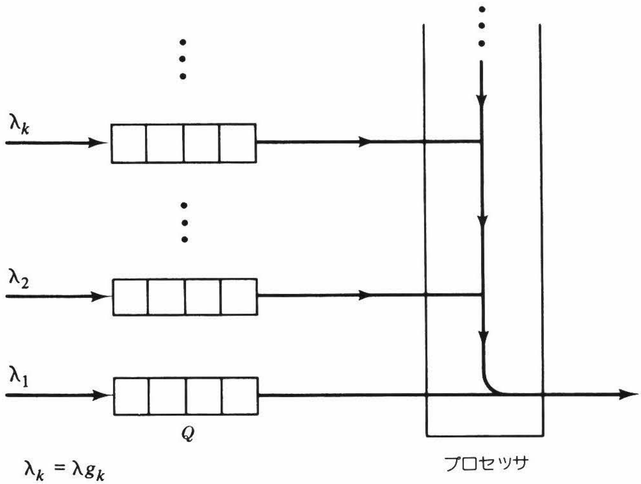



**図 4.7-1**　SRPT システム

り低い待ち行列レベルにジョブが到着したときには，即刻横取り割込みが行われる．

さて，統計的平衡状態において，$kQ$ 単位の実行時間を要するランダム到着のシステム内滞在時間の平均を求めよう．この平均待ち時間は次のように分解される．

(4.7.1)
$$W_k = W'_k + W''_k$$

ここで，$W'_k$ は到着の瞬間から下げ札付きのジョブが初回のサービスに入るまでの平均経過時間であり，$W''_k$ はシステム内で費やす残りの時間の平均である．$W''_k$ は平均**居留**(residence)**時間**ともいわれる．

$W'_k$ を計算するため，SET 規律の解析で行ったのと同じような小細工を弄する．その詳細は次の通りである．**図 4.7-2** に示されている修正システムを考える．そこでは，図 4.7-1 の待ち行列レベルが普通の(横取り割込み不可の)優先権システムを構成するように編成してある．また，待ち行列レベル $i \leq k$ への到着は $i$ 時間量子の実行時間を必要とするが，レベル $k$ よりも高位の待ち行列への到着は全て $k$ 時間量子のサービスしか必要としないと仮定される．したがって，優先権に基づいて行われるレベル 1 から $k$ までの待ち行列に対するジョブの順序付けは SPT による順序付けと同じことになる．

この新システムを吟味すると，実行に $k$ 時間量子を要する到着の**行列にお**

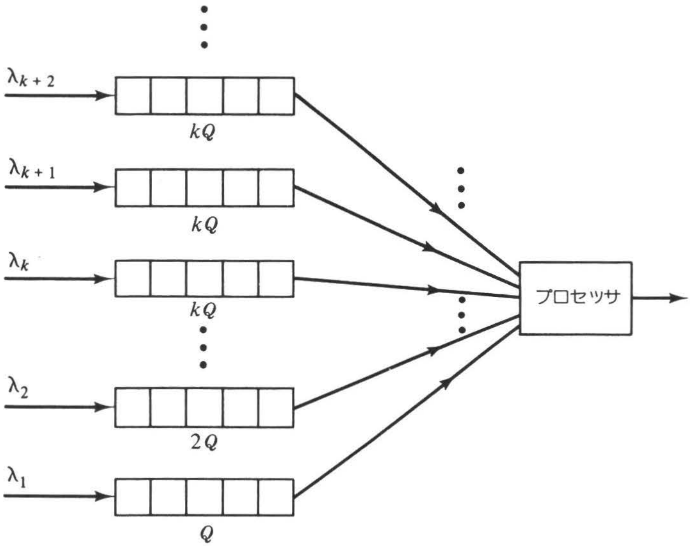



**図 4.7-2** $W'_k$ の計算のための優先権システム

**ける**待ち時間の平均は，正に $W'_k$，すなわち下げ札付きのジョブが元のシステムにおいて最初の時間量子のサービスを受け始める時刻までの待ち時間の平均に他ならないという所見に到達する†．実際，元のシステムの割込みについての規則によれば，行列における最初の待ち状態に入っている下げ札付きのジョブに対してより低い優先権を持つジョブが与える影響は，このジョブがあたかも $k$ 時間量子のサービスしか必要としないジョブとして到着していたかのようである．また，もともと $k$ 時間量子以下のサービスしか必要としないジョブが行列における最初の待ちの状態にある下げ札付きのジョブに対して与える影響は，同類の離散的 SPT システムにおける(下げ札付きのジョブが実行に入るのを妨害するという)影響と全く同じである．したがって，$W'_k$ を計算するためのこの新システムの解析は，4.5 節の一行一行を正確になぞればそれで十分であり，次の結果を齎す．

† **訳注** $k$ よりも高いレベル $l$ の待ち行列の取扱いを吟味するならば，この修正システムでは，下位のレベルの待ち行列が空のとき，レベル $l$ のあるジョブの終了後直ちに同レベルの次のジョブの実行を開始し得ることが分る．しかし，元のシステムの動作を忠実に模倣しようとするならば，レベル $l$ のジョブの終了と同レベルの次のジョブの開始の間に $(l-k)Q$ 時間の空きを置かなければならない．けれども，レベル $k$ のジョブの待ち時間の平均を考える限り両システムは等価であると本文は述べている．

$$W'_k = \frac{\mathrm{E}'(S_r)}{[1 - \lambda H_k(S)]\,[1 - \lambda H_{k-1}(S)]} \tag{4.7.2}$$

ただし,

$$H_k(S) = \sum_{i=1}^{k} (iQ)g_i \tag{4.7.3}$$

また, 上式中の $\mathrm{E}'(S_r)$は, 到着時に進行中(存在するならば)のジョブを終了させるのに要する平均の時間である. $\mathrm{E}'(S_r)$を計算するに当っては, 実行中のジョブに二つのクラスがあることを考慮しなければならない. すなわち, もともと $i \leq k$ 時間量子のサービスしか必要としないものと, 元の SRPT システムにおいては $k$ 時間量子よりも多くのサービスを必要としていて, レベル $k$ よりも上の位置にある待ち行列から来たものである. したがって, 下げ札付きのジョブの到着時に進行中のジョブは, そのクラスに応じて $i$ 時間量子あるいは $k$ 時間量子が割当てられている. ところで, 元のシステムにおいて(残りの)サービス必要量が $k$ 時間量子にまで減少してしまったジョブの発生(これを到着であると言ってもよい)過程はポアソン的ではないが, その発生(到着)率だけは, 修正システムに対する $\mathrm{E}'(S_r)$の計算を行うために知る必要がある. ここで, 式(4.2.29)を修正システムに対して適用すると, 次の結果が得られる.

$$\mathrm{E}'(S_r) = \frac{\lambda}{2}\left[\sum_{i=1}^{k} (iQ)^2 g_i + (kQ)^2[1 - G(k)]\right] = \frac{\lambda \mathrm{E}_k(S^2)}{2} \tag{4.7.4}$$

$W'_k$ が決定したので, 次に平均居留時間 $W''_k$ を計算しよう. 下げ札付きのジョブは1から $k$ までの各レベルでそれぞれ1時間量子を取得しなければならないから, 次式のように書くことができる.

$$W''_k = \sum_{j=1}^{k} (Q + Y_j) \tag{4.7.5}$$

ここで, $Y_j$ は, 下げ札付きのジョブが自己に割当てられた $k - j + 1$ 番目の時間量子を開始する瞬間から完了に到る瞬間までの間にレベル $j$ の待ち行列内において(中断した実行の再開を)待たなければならない時間の平均である. 明らかに, 待ち時間 $Y_j$ は, 下げ札付きのジョブからプロセッサを横取りして割込むより高い優先権を持つ(すなわち, $j$ よりも低いレベルの)ジョブの到着によって生じる. $Y_j$ を計算するには, $Q$ 時間単位の実行の途上にある下げ札付きのジョブを押し退けて割込む到着と $Y_j$ の途中で起こる到着とを分けて考

えるべきである．したがって，次式が成り立つ．

(4.7.6) $$Y_j = \sum_{i=1}^{j-1} (\lambda_i Q) iQ + \sum_{i=1}^{j-1} (\lambda_i Y_j)(iQ)$$

$\lambda_i$ に $\lambda g_i$ を代入すると次のようになる．

(4.7.7) $$Y_j = \lambda H_{j\text{-}1}(S) Q + \lambda H_{j\text{-}1}(S) Y_j$$

ただし，$H_{j\text{-}1}(S)$ は式(4.7.3)によって与えられる． 式(4.7.7)を $Y_j$ について解くと，次式を得る．

(4.7.8) $$Y_j = \frac{\lambda H_{j\text{-}1}(S)}{1 - \lambda H_{j\text{-}1}(S)} Q$$

この式を式(4.7.5)に代入すると次式が得られる．

(4.7.9) $$W''_k = \sum_{j=1}^{k} \frac{Q}{1 - \lambda H_{j\text{-}1}(S)}$$

式(4.7.2)と(4.7.9)を加えると，最終結果が得られる．すなわち，

(4.7.10)
$$W_k = \frac{\lambda \mathrm{E}_k(S^2)}{2[1 - \lambda H_k(S)][1 - \lambda H_{k\text{-}1}(S)]} + \sum_{j=1}^{k} \frac{Q}{1 - \lambda H_{j\text{-}1}(S)}$$

連続的サービス時間分布 $B(x)$ を仮定したときの SRPT 規律の下における実行時間 $t$ のジョブのシステム滞在時間の平均は次の通りである．式(4.7.10)の極限 $Q \to 0$ を取ることにより，一般的結果

(4.7.11)
$$W(t) = \frac{\lambda \int_0^t x^2 dB(x) + \lambda t^2 [1 - B(t)]}{2\left[1 - \lambda \int_0^t x dB(x)\right]^2} + \int_0^t \frac{dx}{1 - \lambda \int_0^x y dB(y)}$$

および，$V(t) = W(t) - t$ が得られる．

本節の結果は Miller と Schrage[25] によって得られた．

## 4.8 処理時間に関する情報に基づく優先権諸規律の比較[8,26]

計算機操作の諸々の確率モデルの研究に応用されてきた解析の諸技法が，こ

れまでのいくつかの節において詳細にいたり述べられてきた．特に，ジョブの処理時間に基づいて優先権を決定するサービス規律が論議の焦点であった．本節では，これらの規律の相対的な利点を論述し，それらに最もよく合ったシステム性能評価の諸規準を特定することを目的とする．

まずは，基本時間間隔，すなわち時間量子を零に近づけたときの極限の場合として得られる諸規律に議論を限定しよう．これらの規律の一般的性質は，適度に小さい量子，すなわち処理時間に比較して小さい量子を持つときの非零時間量子システムの一般的性質を反映しているに違いないからである．見られる通り，これらの規律を特徴づけるのは大変に容易である．

最初に，前節で解析を行ったばかりの最短残存処理時間(SRPT)規律を取上げよう．SRPT 規則では処理時間について完全な情報を持つことが仮定されるから，SRPT による順序付けが退去間隔の最小化を実施できる最良のものを表現しているということは直観的に明らかである．したがって，これもまた直観的ではあるが，SRPT 規律がシステムにおける待ち時間の最小の平均を与える規則を内包しているともいえることになる．このことは，実際，一般到着分布および一般サービス時間分布に対して，全到着時刻が予め知られている場合でさえも証明されている．したがって，リトルの公式を適用するならば，平均のシステム内個数もまた最小になるという結果が得られることになる．このことは，一般に，ジョブを収容するために要する記憶領域の大きさを最小化しようとする傾向を齎すであろう．しかし，記憶領域の必要量について正確な言明を望むならば，ジョブの大きさと実行時間の間の相関関係を(もしそれが存在するならば)考慮に入れなければならないであろう．

SRPT 規則の限界は，横取り割込みの費用が可能性として大であること，特に CPU と補助記憶装置の間でのジョブの転送を含むときに失費が大であるという点にある．そのため，平均待ち時間がある程度増加するという代償を支払う代りに割込みの発生率が減少するような SRPT 規則の種々の変化形も研究されている[14]．もしも，極端に，横取り割込み不可の優先権規律に限ってしまわねばならないとしたならば，丁度 SRPT 規則が全ての規律のクラスにおいて演じたような役割を，今度は，4.5 節で述べた SPT 規則がこの種のサブクラスにおいて演じていることになる．(SPT による順序付けは，決定性モデルに対しても平均システム内滞在時間（平均流れ時間）の最小値を齎すことが 3.8 節の系 3.1 で示されていることを想起しよう．)

SRPT 規律と SPT 規律は，処理時間についての情報が不完全，すなわち処理時間の見積りだけをあらかじめ知ることができるに過ぎないと仮定される場

合に対しても解析されている[8]. 処理時間の見積りが大雑把なものであったとしても平均待ち時間が大幅に減少するという事実は，このような規則を適用することが可能であるシステムをさらに広範囲に拡大してもよいことを物語っている. 平均の流れ時間に対する種々の処理時間依存規律(processing-time-dependent discipilne)の効果に関する興味ある定量的研究が Conway その他[8]によって行われている.

PS 規則と SET 規則によって達成される目標はやや微妙であるので記述しにくい. SRPT 規律あるいは SPT 規律と異なって, PS 規律と SET 規律のどちらにおいても，処理時間があらかじめ知られていないことが仮定されている. 計算機システムをモデル化するに当ってこの仮定は通常必要とされるものである.

処理時間について事前の知識が持てないとすれば，ジョブの処理に要する時間の期待値は処理の経過時間の非減少関数であるということにまず着目することによって SET 規則の特徴付けができるであろう. そのように考えると，SET 規律は，システム内に現に存在しているジョブの中で (経過時間に基づいて) 全処理時間の期待値が最小であると評価されるような1個または複数個のジョブを次にスケジュールするという意味でより短時間のジョブを優遇していることが分る. 一方， PS システムでは，スケジューリングの決定を行う際に利用されていることが明示できる(静的な，あるいは経時変化する)ジョブ特性は全く存在しない. したがって，どのような差別的取扱いも PS 規則には見られない. すなわち，プロセッサの能力が，全てのジョブに対して——それらの間の相対的な特性の違いに関係なく——平等に分配されるようになっている. それにもかかわらず，通常比較の標準として用いられる FIFO 規則と較べれば，PS 規律がより短時間のジョブを平均的により速く通過させてしまうということも事実である. FIFO 規則によれば， 待ち行列内で最も長時間待ったジョブがスケジュールされるようになっている. その結果，FIFO システムでは，短時間のジョブは長時間のジョブによる妨害を，差別的取扱いが全く見られない PS システムにおけるよりも多く受けることになる. 纏めると，PS システムにおけるジョブは，いかなる瞬間でも**残存**処理時間の最も少い順に従って完了することになり， 一方，SET システムにおけるジョブは，いかなる瞬間でも**全**処理時間の最も少い順に従ってシステムから退去することになると述べることによって PS 規律と SET 規律の間の差異を明確にすることができる.

サービス時間の分布が上述の諸規律の性能に影響を及ぼしているということを認識することは重要である. サービス時間が指数分布に従う場合には, 処理時

間についての情報を利用してはいないような規律の下では，それがいかなる規律であっても，システム内滞在時間の平均は異なることがない．したがって，この場合に対しては，SET 規則の下であっても PS 規則の下であっても，システムにおける待ち時間の平均は変らない(ただし，待ち時間のより高次のモーメントには変化が生じる)．サービス時間がもっと一般の分布に従う場合には，システム内滞在時間の平均は**サービス時間の変動**(variation in service times)† によって，影響される．PS 規律および SET 規律の下では，サービス時間の変動が微小になると，性能が悪くなることは直観的に明らかである．極端な場合を考えよう．変動が零である(全てのサービス時間が互いに相等しい)とき，SET 規則の下では全稼働期間が終了するまでのどのジョブも完了しない．これに対し，FIFO 規則の下では1個のジョブを除いて残りの全てのジョブが(全稼働期間の終了よりも)もっと早く完了するということを注意しておく．一方，変動が極めて大であるとき，たとえば，(1秒といった)非常に短時間のジョブと(1時間といった)非常に長時間のジョブの二種類だけがあるとき，PS 規律および SET 規律の下では短時間のジョブは速やかに完了することになるが，FIFO 規律の下では短時間ジョブが長時間ジョブによって著しく遅延させられるかも知れないということは明らかである．

PS 規則に対しては，上記の所見を以下のようにもっと厳密に述べることが可能である．PS システムに対して式(4.4.17)で与えられた待ち時間の平均

$$W_{\mathrm{PS}} = \frac{\mathrm{E}(S)}{1-\rho}$$

と FIFO システムに対して式(4.2.33)で与えられた結果

$$W_{\mathrm{FIFO}} = \mathrm{E}(S) + \rho\frac{[1+C^2(S)]}{2(1-\rho)}\mathrm{E}(S)$$

の差を取ると，次式のようになる．

$$W_{\mathrm{FIFO}} - W_{\mathrm{PS}} = \frac{\rho[C^2(S)-1]}{2(1-\rho)}\mathrm{E}(S)$$

上式から，相対性能の変動係数 $C$ への依存性がはっきりする．指数分布に対しては，$C(S)=1$ であるので，前述したように $W_{\mathrm{FIFO}} = W_{\mathrm{PS}}$ である．サービス時間の変動がもっと大である($C(S)>1$)ときは，PS 規則の下でのシス

† **訳注**　変動は，定量的には変動係数(式(4.2.22)参照)によって表される．

テム内滞在時間の平均は(FIFO 規則の下におけるそれ)より小となり，また，サービス時間の変動がもっと小である($C(S) < 1$)ときは，反対に PS 規則の下でのそれは(FIFO規則の下でのそれ)より大となる†. SET 規律に対しては，同様の結果は求められていない．しかし，サービス分布が‘極端な場合’を考えることにより，下記の式がおそらく成立するように思われる．

$$W_{\mathrm{SET}} < W_{\mathrm{PS}} \quad (C > 1)$$
$$W_{\mathrm{SET}} > W_{\mathrm{PS}} \quad (C > 1)$$

以上を総合すると，PS 規律と SET 規律には，(他のジョブに混って，より短い処理時間を持つジョブが存在するとき，)より短い処理時間を持つジョブの処理を促進せしめるという有利な点がある一方で，サービス時間のある(特に，FIFO 規律を標準に取った場合には変動係数が 1 より小であるような)大きなクラスに対しては応答時間が(FIFO 規律の下でのそれよりも)劣悪になるという不利な点も存在している．なお，これらの規律については，数値的研究が Coffman と Kleinrock[13]および Schrage[14]によって行われている．

## 演 習 問 題

4.1 過程 $X(t)$の周期とは，1 遊休期間とそれに続く全稼働期間から成る時間間隔であると定義する．統計的平衡状態においては，M/G/1 型システムの 1 周期内の平均到着数が 1 全稼働期間内にサービスされる平均個数に等しいということは一見して分る．このことを用いて，平衡状態における全稼働期間の長さの平均 $\mathrm{E}[D]$は次式のように表現できることを示せ．

$$\mathrm{E}(D) = \frac{\mathrm{E}(S)}{1 - \lambda \mathrm{E}(S)}$$

この結果を，式(4.2.46)から 1 次モーメントを計算することによっても確かめよ．また，次式をも算出せよ．

$$\mathrm{E}(D^2) = \frac{\mathrm{E}(S^2)}{[1 - \lambda \mathrm{E}(S)]^3}$$

4.2 到着は集団で行われるとし，また到着時間間隔はパラメータ $\lambda$ を持つ指

† **訳注** 次のように式で表した方が見やすい．

$$W_{\mathrm{PS}} < W_{\mathrm{FIFO}} \quad (C > 1)$$
$$W_{\mathrm{PS}} > W_{\mathrm{FIFO}} \quad (C < 1)$$

数分布に従うと仮定する(これがいわゆる**集団到着**(bulk arrival)機構である). 到着集団の大きさが $k$ である確率は, 定常的であり, $g_k$ で与えられるとする. また, $G(z) = \sum_{k=1}^{\infty} g_k z^k$ はそれに対応する母関数であるとする. 4.2節の記号を用いて, 次式が成り立つことを示せ.

$$A(z) = B^*(\lambda - \lambda G(z))$$

式(4.2.16)で示された $P(z)$ の表現式に上式を代入すると, 一般のサービス時間分布を持つ集団到着待ち行列に対する $\{\pi_i\}$ の解が得られることを示せ.

4.3 **横取り割込み-再開**(preemptive-resume)規則を用いると, 4.5節の優先権規律は次のように修正される. より低い優先権のジョブがプロセッサ上にあるときにより高い優先権のジョブが到着すると, 直ちに, 前者はプロセッサ上から排除されて元の待ち行列の先頭に戻され, 代りに高い優先権のジョブがサービスされ始める. 割込まれたジョブが後になってプロセッサ上に戻るときには, 割込みによって中断した所から実行を再開する. すなわち, 割込まれたために サービス時間を無駄にすることはない. 4.5節における定義と記号を使って, システム内滞在時間の平均が次式で与えられることを示せ.

$$W_k = \frac{\mathrm{E}(S_k)}{1-\beta_{k-1}} + \frac{\sum_{i=1} \lambda_i \mathrm{E}(S_i^2)}{2(1-\beta_{k-1})(1-\beta_k)}$$

(4.5節~4.7節で解説した方法を用いよ.)

4.4 どの時点でもプロセッサは残存処理時間の期待値が最小であるジョブ(あるいは, プロセッサ共有形態では複数個のジョブ)を実行していることになるようなサービス規律を定義せよ. ただし, 上記の期待値はプロセッサ上でジョブが既に受けた時間の量だけに基づいて計算される[26]. 互角のものに対しては FIFO 規則を行使する. サービス時間分布を $B(x)$ とし, またポアソン到着を仮定せよ.

上記の規律が FIFO 規律に縮退してしまうのは, $B(x)$ に関するどのような条件の下であるか. また, 帰する所 SET 規律になってしまうのはどのような条件の下であるか. これらの条件を満たす分布の実例を上げよ. さらに, この規律が FIFO 規律のようにもまた SET 規律のようにも作動しないような分布をも示せ.

**4.5** 新たな到着は行列が長いと列に加わる意欲が減退するという事実をモデル化したいとする．この目的で，$\lambda_n = \lambda/(n+1)$を仮定して，式(**4.3.5**)および(**4.3.6**)に従うようなポアソン待ち行列を定義しよう．もし$\mu$が指数サービス時間分布のパラメータであるならば，システムの長さの定常分布はパラメータ$\lambda/\mu$を持つポアソン分布になることを示せ．

**4.6** 4.2節のもう一つの例解として，次のような2-レベルの優先権待ち行列を調べよう．高優先権待ち行列へは到着率$\lambda$のポアソン入力があり，一方，低優先権待ち行列は，平均値が$1/\mu$の指数分布するサービス時間を持つジョブで飽和していると仮定する．高優先権待ち行列のジョブは，1次と2次のモーメントがそれぞれ$\mathrm{E}(S)$と$\mathrm{E}(S^2)$である一般分布関数$B(x)$に従うとする．（参考のためこのシステムを図示すると，**図 P4.6**のようになる.）

高優先権待ち行列が空であるときに高優先権ジョブが到着したとすると，(飽和しているという仮定から必ず)実行中であるはずの低優先権ジョブはそのまま続けて$d \geq 0$時間単位のサービスを受けることが許されていると仮定する．もしそのジョブが$d$時間単位内に完了したならば，高優先権待ち行列は割込みを起こすことなしにサービスを開始する．もし実行中のジョブが$d$時間単位よりも多くの時間を必要とするならば，そのジョブがこの加給された$d$時間単位のサービスを受け終ったときに割込まれて，プロセッサは高優先権待ち行列に引き渡される．したがって，もし$d = \infty$であるならば，横取り割込みのない**行列先頭優先権** (head-of-the-line priority)待ち行列となり，もし$d = 0$であるならば，横取り割込み-再開方式の優先権待ち行列となる．

$\{X_i\}$は，状態を高優先権待ち行列内のジョブ数で定義し，高優先権ジョブの退去直後の時点を再生点とするマルコフ連鎖を表すものとする．$F(t)$は，高優先権待ち行列が空であるときに到着する高優先権ジョブがシステム内で費やす時間の分布を表すものとする．（これはサービス時間と

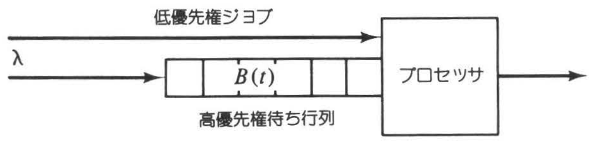



**図 P 4.6** 2-レベル割込み優先権待ち行列システム

サービス開始までの遅延時間の和の分布である.) $F(t)$ の平均を $\mathrm{E}(S')$ で表そう.

$\{X_i\}$ の定常分布の存在条件を決定し, この分布の母関数は次式で与えられることを示せ†.

$$P(z) = \pi_0 \frac{zF^*(\lambda - \lambda z) - B^*(\lambda - \lambda z)}{z - B^*(\lambda - \lambda z)}$$

ただし

$$\pi_0 = \frac{1 - \lambda \mathrm{E}(S)}{1 + \lambda \mathrm{E}(S') - \lambda \mathrm{E}(S)}$$

リトルの公式

$$W = \frac{1}{\lambda} \lim_{z \to 1} P'(z)$$

から平均の流れ時間を計算せよ.

**4.7** 次のような2-レベル(離散的)SPT 規律を考えよう. $L$ 時間単位以下のサービスを要する全てのジョブをクラス1のジョブと呼び, 残りのジョブはクラス2のジョブと呼ぶ. 2-レベルの普通の(横取り割込み不可の)優先権待ち行列(4.5節)を, 高優先権レベルはクラス1のジョブから成り, 低優先権レベルはクラス2のジョブから成るように実現する. 両者を混成した到着機構は到着率 $\lambda$ のポアソン到着になるようにできている. $B(x)$ は, E$(S)$ と $\mathrm{E}(S^2)$ で示される1次と2次のモーメントを持つサービス時間の分布関数を表すものとする.

システム内滞在時間の(両クラスの)総合平均

$$W = W_1 B(L) + W_2[1 - B(L)]$$

は次式で与えられることを示せ.

$$W = \frac{\lambda \mathrm{E}(S^2)}{2(1-\rho)} \left[ \frac{1 - \rho B(L)}{1 - \rho_1} \right]$$

ただし, $\rho = \lambda \mathrm{E}(S)$, $\rho_1 = \lambda_1 \mathrm{E}(S_1)$, ここで, $\lambda_1$ と $\mathrm{E}(S_1)$ はそれぞれクラ

† もっと一般的な見方をすると, $P(z)$ は, 全稼働期間の最初のジョブが $B(x)$ とは異なる分布 $F(x)$ を持つような M/G/1 型待ち行列システムの重要かつ特殊なクラス[27]に対する解になっていることに注意しよう.

ス1のジョブの平均到着率と平均サービス時間である.

$B(x)$は区間$(0, d)$の上の一様分布であると仮定すると，平均システム内滞在時間 $W$ を最小にする $L$ の値は次式で与えられることを示せ.

$$L = \frac{2}{\lambda}\left[1 - \sqrt{1 - \frac{\lambda d}{2}}\right]$$

さらに $B(x)$はパラメータ $\mu$ を持つ指数分布であると仮定すると, $W$ を最小にする $L$ の値は次式を満足することを示せ.

$$\frac{\mu}{\lambda} - \frac{e^{-\mu L}}{\mu L - 1} = 1$$

4.8 図 P4.8 に，**縦列型待ち行列**(tandem queues)システム(直列型待ち行列(queues in series)システムともいう)が示されている. ジョブは, 第1(段)のプロセッサ $P_1$ 上でサービスを受け終わると直ちに第2(段)の待ち行列に加わり， $P_2$ 上での第2 '段階' のサービス受領を待つ. 直列型の待ち行列システムは，計算機システムの研究においてモデルとして使われることが多い. (もっと一般性のある議論は5.4 節で行われる.)

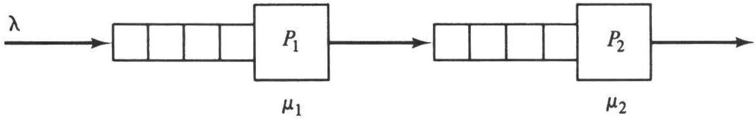



**図 P 4.8**

$P_1$ および $P_2$ 上でのサービス時間はそれぞれパラメータ $\mu_1$ および $\mu_2$ を持つ指数分布に従うものと仮定する. また， 入力は到着率 $\lambda$ を持つポアソン到着である. 平衡状態において $i$ 個が第1段のシステム内に滞在中((第1段の)待ち行列内で待機中であるかまたは(第1のプロセッサ上で)サービス受領中である)であり， かつ $j$ 個が第2段のシステム内に滞在中である確率を $p_{ij}$ で表そう. 4.2 節の方法を用いて $p_{ij}$ の平衡方程式を書き，さらにその解は次の通りであることを示せ.

$$p_{ij} = [(1 - \rho_1)\rho_1{}^i][(1 - \rho_2)\rho_2{}^j] \quad (i = 0, 1, \ldots, \quad j = 0, 1, \ldots)$$

ただし，$\rho_i = \lambda/\mu_i (i = 1, 2)$である. 第1および第2の待ち行列の長さの分布は互いに独立であり，しかも，それぞれパラメータが $\rho_1$ および $\rho_2$ で

ある幾何分布になるという結論が得られる．この結論は，第1の待ち行列への入力がポアソン到着であり，直列に $k$ 段接続された指数型の待ち行列のネットワークに容易に一般化することができる．

第2の待ち行列の振舞は，その待ち行列が到着率 $\lambda$ のポアソン入力を持つときと全く等しいということが上記の結論によって示唆されていることに注意しよう．事実，第1の待ち行列の出力はポアソン的であって，その発生率は $\lambda$ である(演習問題4.11を見よ)．さらに，この所見は，$P_2$($k$ 個の待ち行列を直列に接続したもっと一般の場合には $P_k$)上でのサービス時間が一般分布に従うと仮定しても，M/G/1 型待ち行列に対して4.2節で求めた結果を用いて，解析を行うことが可能であることを示している．

4.9　SRPT システムを考える．サービス時間分布は $B(x)$ とする．$p$ 時間単位のサービスを必要とするジョブがプロセッサを丁度割当てられた所であるとしよう．このジョブが $x \leq p$ 時間単位経過するまでに割込まれる確率を $W_p(x)$ で表す．微分方程式

$$W_p'(x) + \lambda B(p-x)\,W_p(x) = \lambda B(p-x) \quad (0 \leq x \leq p)$$

を導出し，実行に $p$ 時間単位を要するジョブが受ける横取り割込みの回数の分布はパラメータ

$$\alpha = \lambda \int_0^p B(t)\,dt$$

のポアソン分布であることを示せ．

4.10　4.2節に述べたマルコフ連鎖 $\{X_i\}$ の議論から次式のように書くことができる．

(P4.10.1)　$$X_{n+1} = X_n + a_n - \delta_n$$

ただし，$a_n$ は区間 $(t_n, t_{n+1})$ 内の到着数であり，また

$$\delta_n = \begin{cases} 0 & (X_n = 0) \\ 1 & (X_n > 0) \end{cases}$$

である．$a_n$ の確率分布は，式(4.2.17)で示された母関数を持つ $\{q_i\}$ によって与えられている．式(P4.10.1)の期待値を取ることにより次式が成り立つことを示せ．

$$\lim_{n\to\infty}\Pr[X_n > 0] = \rho$$

ただし，$\rho = \lambda\mathrm{E}(S)$である．次に，式(P4.10.1)の両辺をまず2乗し，さらに期待値を取ることにより，極限においては式(4.2.21)に示されている$\overline{n} = \lim_{n\to\infty}\mathrm{E}(X_n)$が得られることを証明せよ．

4.11 到着時間間隔とサービス時間がそれぞれ指数分布する待ち行列の退去時間間隔は，これまた到着時間間隔の分布と同じパラメータを持つ指数分布に従うことを示せ．それには，もし$t$が退去時点の直後の時刻であるならば，次の退去までの時間は，確率$1 - p_0 = \Pr\{X(t) > 0\}$で1サービス期間であり，確率$p_0$で1遊休期間と1サービス期間の和となることに注意する必要がある．

上記の結果は，複数サーバのポアソン型待ち行列にも拡張できる．実際，ポアソン入力を持つ1個またはもっと多くの並列の指数型サーバの出力は，これまたポアソン的である．しかし，これを示すには，引き続く退去の時間間隔が互いに独立であり，同じパラメータを持つ指数分布に従うことを明らかにする必要がある．(たとえば，Takacs[5]を参照せよ．)

4.12 式(4.2.26)は，ランダム到着時に処理が進行中であるジョブの実行時間の密度関数を与えている．ところで，到着時に進行中のジョブは実行に$y$時間単位を要するジョブであるという条件が与えられたとき，残存実行時間が$x$である条件付密度は単に一様分布の密度$1/y(0 < x < y)$に等しい．このことから，(条件付きでない)残存実行時間のラプラス変換$R^*(s)$(4.2.3節を参照せよ)は次式のようになることを示せ．

$$R^*(s) = \frac{1 - B^*(s)}{s\mathrm{E}(s)}$$

4.13 平衡状態にあるLIFO式M/G/1型システムの行列における待ち時間のラプラス変換を求めるために，稼働中のシステムに到着するジョブは，到着時に進行中であるジョブの残存実行時間から開始される全稼働期間の終了まで待たなければならないということに着目しよう．この意味の全稼働期間のラプラス変換は，$B(x)$を$R(x)$で置き換えることによって式(4.2.46)から得ることができる．演習問題4.12の結果を利用して，LIFOシステムでは次式が成り立つことを示せ．

$$V^*(s) = (1-\rho) + \frac{\lambda(1-H^*(s))}{s+\lambda-\lambda H^*(s)}$$

ただし，$H^*(s)$はM/G/1型システムの全稼働期間分布のラプラス変換である． FIFO および LIFO に対して行列における待ち時間の2次モーメントをそれぞれ計算し，結果を比較せよ．

4.14 計算機システムでは，いくつかの相異なるサービス設備(記憶装置，I/O チャネル等)が1個の有限な待ち合わせ設備を共用しなければならない場合が時折見られる． 図 **P 4.14(a)**に示されているようなその種の簡単なシステムを考えよう．そこでは，$P_1$ 上で実行しなければならない実行単位(unit)と $P_2$ で実行しなければならない実行単位がそれぞれ到着率 $\lambda_1$ と $\lambda_2$ のポアソン流れを成して到着するようになっている．共用される待ち行列は $N$ 個の場所からなるものとし， $P_1$ と $P_2$ に対してはそれぞれパラメータ $\mu_1$ と $\mu_2$ の指数サービス時間分布が仮定される． このシステムを解析し， 処理率(throughput)と待ち行列内で待っている実行単位の二つのク

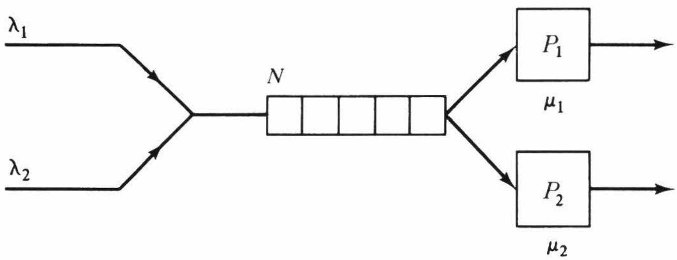



図 **P 4.14(a)**

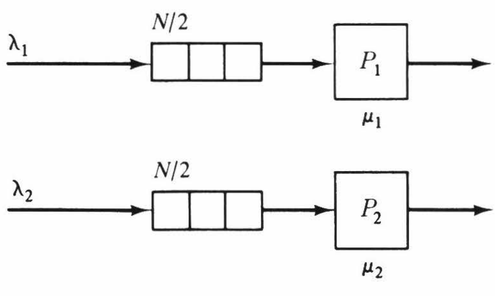



図 **P 4.14(b)**

ラスの間の相互干渉の尺度を示せ．（なお，待ち行列が一杯であるときに到着した実行単位は行列に参入できずに‘失われ’てしまうと仮定せよ.）

上述のシステムの性能を，$N$は偶数であると仮定して，各々が$N/2$個の場所を持つ互いに分離された2個の待ち行列システムと比較せよ．すなわち，上述のシステムを**図 P4.14(b)**に示されているシステムと比較せよ．

4.15 ポアソン(M/M/$r$)型待ち行列の理論は，**段階法**(method of stages)を用いて，指数分布に従わないサービス時間または到着時間間隔を伴う多くのシステムに適応するように拡張することができる．本問題と次の問題はこの拡張を扱う．到着率が$\lambda$のポアソン入力を持つシステムが直列に接続された$k$個の‘段階(stage)’(それぞれがパラメータ$\mu$を持つ互いに独立な指数分布に従う)，によって表現されるようなサービス時間分布を持つものと仮定しよう．和の分布を$B_k(x)$で表すことにし，それがラプラス変換$B^*_k(s) = \mu^k/(\mu + s)^k$を持つはずであることに着眼せよ．まず，その変動係数が次式

$$C_k(S) = \frac{1}{\sqrt{k}}$$

で与えられることを確かめ，したがってまた，$B_k(x)$は指数分布よりも小さい変動係数を持つ分布をモデル化したものであると考えられることを確かめよ．($k$が無限大になると，$B_k(x)$は一定値のサービス時間を与える分布になるということに注意しよう.)

ポアソン型待ち行列の理論を適用するために，ここでは，単一サーバの場合のシステムの状態を，システムが現在1個のジョブを段階$p$でサービス中であり，しかも待ち行列中には$q$個が存在しているならば，この状況に対応する一連番号$r = p + kq$によって表現する．システムの状態の平衡状態確率を$p_i$で表し(ただし，$j < 0$ならば，$p_j = 0$と定義する)，まず次の式が得られることを示せ.

$$\lambda p_0 = \mu p_1$$
$$(\lambda + \mu)p_r = \mu p_{r+1} + \lambda p_{r-k} \quad (r \geq 1)$$

次に，$p_i$の母関数を求めよ(この問題と演習問題4.2との関係に注意せよ)．なお，システム内に$n$個のジョブが存在する確率は次式から計算できることに注意しよう.

$$\sum_{r=(n-1)k+1}^{nk} p_r$$

上記の段階法を複数サーバ待ち行列に応用することを考えよ．また，互いに独立ではあるが互いに等しいとは限らない指数分布の和に従うサービス時間に対しても直ちに一般化できることも看取するべきである．この場合に対する解析を進めるにあたっては，サービス中の段階を同定できる手掛り(段階番号)を含むように状態変数を増すとよい．

4.16　1より大きい変動係数を持つ分布をモデル化する必要があるときには**並列段階法**(method of stages-in-parallel)を適用することができる．特に，ジョブは確率 $\sigma_i(\sum_{i=1}^{k}\sigma_i=1)$ でパラメータ $\mu_i$ の指数分布に従うサービス時間を持つものと仮定しよう．このとき，ラプラス変換は

$$B^*{}_k(s)=\sum_{i=1}^{k}\frac{\sigma_i\mu_i}{\mu_i+s}$$

となることをまず確かめ，変動係数はパラメータを適当に選べばいくらでも大きくし得ることを示せ．$n$ をシステム内のジョブ数とし，$i$ はサービス中の段階の番号であるとして，状態変数を $(n, i)$ と定めるとき，平衡状態における単一サーバに対する平衡方程式を求めよ．複数サーバ待ち行列と非指数的到着時間間隔を持つシステムへの一般化を考えよ．最後に，直列および並列に接続された指数分布の組合わせに対しても，この方法を拡張できることを確かめよ．（実際，そのような分布によって，ラプラス変換が二つの多項式の比となるような正値の確率変数のいかなる分布でも表現可能である．**一般アーラン分布**(general Erlangian distribution)の族はこれらの分布によって構成される．）

実際問題としては閉じた形の解† が得られる連立方程式が立てられるように一般的なモデルを構成することができると期待するのは無理である．そのため，結果を得るための普通の方法ではまず，連立方程式を（ある点を越す確率を零と置くことにより）有限な一組に減らし，次にこれを必要な精度を得るまで計算過程を繰り返しながら数値的に解いて行くといった手順が取られることになる．その際に，計算時間が許容される範囲内に収まるように，モデル化に含まれる構造パラメータの数を可能な限り少なくし

† **訳注**　閉じた形の解というのは，陽関数の形で表示された解をいう．

ておかなければならないことは明らかである．

## 参考文献


1. FELLER, W., *An Introduction to Probability Theory and Its Applications* (Vol. I). Wiley, New York (1950).
2. COX, D. R., and W. L. SMITH, *Queues*. Methuen, London (1961).
3. MORSE, P. M., *Queues, Inventories, and Maintenance*. Wiley, New York (1958).
4. SAATY, T. L., *Elements of Queueing Theory*. McGraw-Hill, New York (1961).
5. TAKACS, L., *Introduction to the Theory of Queues*. Oxford University Press, Inc., New York (1962).
6. RIORDAN, J., *Stochastic Service Systems*. Wiley, New York (1962).
7. PRABHU, N. U., *Queues and Inventories*. Wiley, New York (1965).
8. CONWAY, R. W., W. L. MAXWELL, and L. W. MILLER, *Theory of Scheduling*. Addison-Wesley, Reading, Mass. (1967).
9. LITTLE, J. D. C., "A proof for the queueing formula $L = \lambda W$." *Opns. Res. 9*, 3 (May 1961), 383–387.
10. SAKATA, M., S. NOGUCHI, and J. OIZUMI, "Analysis of a processor-shared queueing model for time-sharing systems," in *Proc. 2nd Hawaii Int'l Conf. Sys. Sci.*, Univ. of Hawaii, Honolulu (1969), 625–627.
11. SCHERR, A. L., "An Analysis of Time-shared Computer Systems." Ph.D. thesis, Dept. of Electrical Engineering, M. I. T., Cambridge, Mass. (June 1965).
12. KLEINROCK, L., "Analysis of a time-shared processor." *Nav. Res. Log. Quart. 11*, 10 (Mar. 1964), 59–73.
13. COFFMAN, E. G., JR., and L. KLEINROCK, "Feedback queueing models for time-shared systems." *J. ACM 15*, 4 (Oct. 1968), 549–576.
14. SCHRAGE, L. E., "Some queueing models for a time-shared facility." Ph.D. thesis, Cornell University (Feb. 1966).
15. KLEINROCK, L., "Time-shared systems: A theoretical treatment." *J. ACM 14*, 2 (Apr. 1967), 242–261.
16. MUNTZ, R. R., "Waiting time distribution for round-robin queueing systems," in *Proc. Symp. Comp.-Commun., Networks, Teletraffic*. Microwave Research Institute, Polytechnic Institute of Brooklyn (Apr. 1972).
17. COFFMAN, E. G., JR., R. R. MUNTZ, and H. TROTTER, "Waiting time distributions for processor-sharing systems. *J. ACM 17*, 1 (Jan. 1970), 123–130.
18. COFFMAN, E. G., JR., "Stochastic models of multiple and time-shared computer operations." Ph.D. thesis, Univ. of California at Los Angeles (June 1966).

19. Krishnamoorthi, B., and R. C. Wood, "Time-shared computer operation with both interarrival and service time exponential." *J. ACM 13*, 3 (July 1966), 317–338.

20. McKinney, J. M., "A survey of analytical time-sharing models." *Computing Surveys 1*, 2 (1969), 105–116.

21. Cobham, A., "Priority assignment in waiting-line problems." *Opns. Res. 2* (Feb. 1954), 70–76.

22. Kesten, H., and J. Th. Runnenberg, "Priority in waiting-line problems." *Proc. Akad. Wet. Amst., Ser. A, 60*, 3 (1957), 312–324 (Pt I), 325–336 (Pt II).

23. Phipps, T. E., "Machine repair as a priority waiting-line problem." *Opns. Res. 9* (Sept.–Oct. 1961), 732–742.

24. Schrage, L. E., "The queue M/G/1 with feedback to lower priority queues. *Manag. Sci. 13* (1967), 466–474.

25. Miller, L. W., and L. E. Schrage, "The queue M/G/1 with the shortest remaining processing time discipline." *Opns. Res. 14* (1966), 670–683.

26. Coffman, E. G., and R. R. Muntz, "Models of resource allocation using pure-time-sharing disciplines." *Proc. 24th ACM Natl Conf.* (1969), 217–228.

27. Welch, P. D., "On a generalized M/G/1 queueing process in which the first customer of each busy period receives exceptional service," *Opns. Res. 12* (Sept.–Oct. 1964), 736–752.

# 補助記憶および緩衝記憶のモデル

# 5

## 5.1 はじめに

多重プログラミング・システムにおいて総合的性能を決定付ける主な要因は，主記憶と補助記憶との間で行うことができる情報転送の速度であることがしばしばである．この**転送速度**(transfer rate)は，単に物理的な装置の特性だけではなく，情報に対する諸要求を順序付ける手続きの関数でもある．本章の主要な目的は，このような記憶サブシステムを解析するための諸技法を考えることであり，またそれらの性能を最大化するための諸手続きを検討することである．現用のほとんどのシステムにおいては，ディスクあるいはドラム・システムが補助記憶の役割を占めている．それゆえ，本章では，ドラムあるいはディスク様式の記憶装置に関心が注がれることになる．読者はこれらの装置の基本構成(第1章参照)によく通じているものと仮定される．設計の細部の知識，特に全てのディスクまたはドラム・システムに共通しているのではないような知識は，一般にここでの取扱いを理解する上で必要とはならないはずである．

ドラムあるいはディスクのような回転式記憶装置は，その動作特性によってこれらの装置の用法に適用し得る最適化技法を決定付ける．これらの二種の装置の間で本質的により単純なのはドラムである．なぜかというと，ドラムでは，装置の状態を記述するのにただ一つの状態変数だけが必要となるからである．すなわち，時間の関数としてのドラムの角位置は装置の物理的な**瞬間構図**(configuration)を完全に記述する．それに対して，ディスク装置では，その物理的な瞬間構図を記述するのに二つの状態変数が必要である．すなわち，ディスクでは，角位置は状態変数の一つであり，もう一つの状態変数としてヘッドの位置が必要である．このことから，ドラム操作† の特性がディスク操作の特

† **訳注** 原文は 'drum operation' と 'disk operation' である．本章では，ハードウェアとしてのドラムとディスクの '動作' 特性だけを問題としているのではなく，これらのハードウェアの制約の下にあってサービスに対する諸要求を順序付ける手続きの特性を特に問題にしているのであるから，その意を汲んで，'ドラム操作' および 'ディスク操作' という語を用いた．

性の一部を構成することは明らかであろう．それゆえ，やがてはっきりするように，ドラムの利用効率を最適化するための手続きはディスク・システムにも適用することができることになる．

前に示唆したように，個々のドラム・システムあるいはディスク・システム(特に，ディスクの場合)は，(総合的な)性能に支配的な影響を及ぼすことがあり，しかもこれらの装置のクラスに共通しているのではないようなどちらかと言えば特殊化された造りとか制約を持つことが往々にしてある．たとえば，**記録間隙**(recording gap)，ヘッド切換え時間のある特殊な様相，ディスク上の情報の特殊な編成等の存在がそれである．本章の目標は応用範囲の広い解析的接近法を用いて問題を一般的に取扱うことにあるのであるから，通例に依って，これらの装置の抽象的なモデルを定式化することにより，上述の‘特殊な諸事例’にかかずらうことは避けることにしよう．このような抽象的モデルの研究に密接に関係している一般的諸技法は，通常，適切な特殊化と理想化を施すことによって実在のシステムに適用することが可能になる．

待ち行列における遅れを除外すれば，補助記憶と主記憶の間の転送を完了するまでに要する時間には三つの主成分がある．すなわち，1)**探索時間**(seek time)――アームの位置決めによる遅れ(arm positioning delay)――，2)**回転待ち時間**(rotational latency)，および3)**転送時間**(transfer time)である．転送時間は一般にオペレーティング・システムの設計者によって左右されるようなものではないし，また，これは回転待ち時間の終了に続いて直ちに生起するものである．このような事情から，ここでのモデルは，**図5.1-1**に示されてい

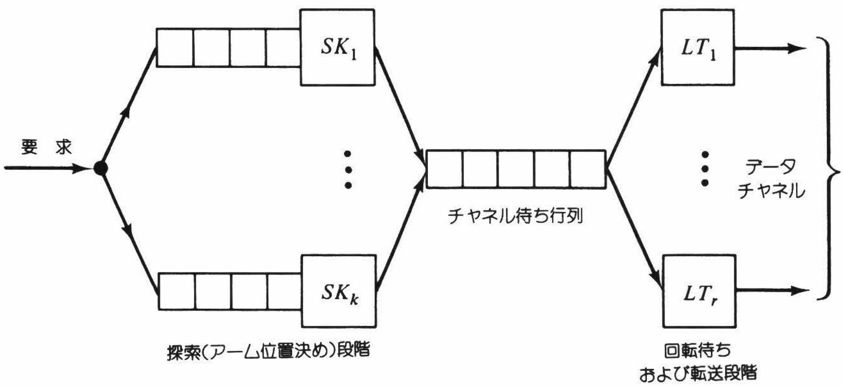



**図5.1-1**　一般モデル

るように，二つのサービス段階，すなわち，1)**探索**(SK)**段階**(seek phase)，2)**回転待ちおよび転送**(LT)**段階**(latency-and-transfer phase)を表す．一般の場合を想定すれば，複数のドラム装置または複数のディスク装置と複数の独立したアーム位置決め機構が集中制御の下にあって，それぞれを並列に操作することが可能であろう．この並列性と連動することにより，複数のデータ・チャネルがLT段階の並列性を可能にする．しかし，普通は，与えられた要求に対して一度あるディスク・アームが選択されて位置決めされてしまうと，そのアームはこの与えられた要求に対応するLT段階が遂行されてしまうまで次の要求に対して使用可能にはならないことを仮定しなければならない．したがって，この仮定の下では，$k$ 個のシーク機構から成る有限入力源がLT段階に対して存在することになる．参考文献[1-6]では，大抵の現用ディスク・システムの設計に付き物の種々の並列性が記述されている．図5.1-1において並列に接続されている待ち行列は，解析の目的上，一般には互いに独立であると仮定される．たとえこの仮定が現実には成立しないとしても，それが解析を扱い易くしておく唯一の手段であるからである．いずれにせよ，独立性の仮定はただ一つの待ち行列に注意を集中することによって全体的な性能の尺度を得ることをしばしば可能ならしめる．この独立性の仮定が与えられたとしても，図5.1-1に示されている一般モデルの解析は，依然として無視し得ない困難，特に，後続のいくつかの節で解析されることになっているより効率的な(かつ数学的には複雑な)順序付けの諸規則の問題に直面している．なお，データ・チャネルが唯一本存在すること，諸分布が指数分布であること，順序付け規則はFIFO式であることを仮定したときの図5.1-1のモデルに対する近似的解析が文献[4]で行われている．

図5.1-1に一般化されて示されているディスク・システムに現出する縦列待ち行列(tandem queues)(または直列待ち行列(queues in series))をより簡単に取扱うために普通に行われる単純化は，探索段階の出力がポアソン的であると仮定することである．(演習問題4.11を見よ.)FifeとSmith[1]，およびAbateその他[3]は，この近似について論じ，動機付けを行った．一方，SK待ち行列が飽和状態に置かれていると仮定されたときには別の単純化が行われることを意味する．この場合には，SK段階とLT段階に対してそれぞれ指数分布を仮定すると，4.3節および4.4節で解析を行った有限入力源を持つポアソン型待ち行列に帰着することになる．上述のような形の近似がいつでも必要であるということではない．ある種のシステムでは，図5.1-1中の待ち行列が全て存在するという訳ではないからである．特に，ある一つの要求が処理を開

始したとすると，その要求は，次の要求が第一の段階で処理を開始する以前に，二つのサービス段階を通してすっかり処理されてしまわなければならないような場合もあるかも知れない．

単純に探索段階を除去することにより，図 **5.1-1** はドラム・システムのモデルであると見なすことができる．次の二つの節では，SK 段階と LT 段階を分離して考察する．目的は，ディスク・ヘッドの動きと回転待ち時間をそれぞれ減少させることによって性能を最大ならしめるように設計された現存するかもしくは提案されている順序付け手続きを特徴付けることである．ドラムとディスクの双方に適用可能な回転待ち時間の減少化がまず考察される．解析的な結果は様々な順序付けアルゴリズムに対する洞察を齎すとはいうものの，現実のシステムを評価するためには，さらに多くの細部を考慮に入れなければならない．数値的研究とシミュレーション的研究に関しては，参考文献，特に Teorey と Pinkerton [6] を参照するとよいであろう．

**5.4** 節では，補助記憶と入出力システムについてさらに広い見方が取られる．ほとんど任意の構造を持った多重プロセッサ・システムを表現する一般待ち行列ネットワークモデルが指数分布を仮定して解析される．入出力プロセッサのシステムから抽出した実例も調べられる．

緩衝領域設計問題の解析を **5.5** 節で行って本章を締括る．Knuth [7] は，FIFO，LIFO，およびこれらの規律の組合せを用いた待ち行列の記憶技法の詳細な取扱いを提供している．同節では，**記憶領域利用率** (storage utilization) の確率モデルおよびタイムシェアリング・システム等の応用分野で発生する実時間低速データ伝送緩衝領域問題に重点が置かれる．

## 5.2　回転待ち時間の最小化

図 **5.1-1** において LT 段階が **1** 個から成っている場合，すなわち，$r=1$ と仮定される場合を考察しよう．この待ち行列をサービスする最も簡単な方式は FCFS (first-come-first-served) 規律である．$N$ はドラム[†] の円周に沿って記憶可能な総語数を表す．待ち行列に順次到着する要求は，系列 $a_1, \ldots, a_i, \ldots,$ および $\tau_1, \ldots, \tau_i, \ldots$ で記述される．ただし，$a_i$ は開始番地の角位置を示し，$\tau_i \geq 1$ は $i$ 番目の要求を転送するのに要する時間を表す．次の諸仮定から基本モデルが得られる．

† 結果はディスクの回転待ち時間問題にも適用されるが，便宜上，ドラム・システムの用語を使うことにする．

1. 到着率 $\lambda$ のポアソン到着が行われる.
2. 系列 $\{a_i\}$ および $\{\tau_i\}$ は互いに独立であり，各々が独立な確率変数の系列である.
3. どの $i$ に対しても，$\{1, 2, \ldots, N\}$ に属する全ての開始番地が等確率で $a_i$ の値となり得る.

このモデルで $N$ の値が適当に大きいとすると，$i$ 番目の要求のサービス時間の期待値は近似的に $T/2+\tau_i$ となる．ただし，$T$ はドラムの1回転に要する(一定値の)時間である．今，$\tau_i$ の分布が1次モーメント $\mathrm{E}(\tau)$ および2次モーメント $\mathrm{E}(\tau^2)$ を持つものとする．平均行列長については，$\rho=\lambda[\mathrm{E}(\tau)+T/2]$ および $\mathrm{E}(S^2)=\mathrm{E}[(\tau_i+\theta)^2]$ として式(4.2.21)を利用することができる．ただし，$\theta$ は，ドラムが回転して開始番地に到達するまでの遅延時間を表す確率変数である．適当に大きい $N$ に対して，$\mathrm{E}(S^2)$ は $\mathrm{E}(\tau^2)+T\mathrm{E}(\tau)+T^2/3$ によって極めて密に近似される．この式はドラム円周上の開始番地の連続分布に対応している．したがって，次式が成り立つ.

$$\bar{n}\approx\lambda\left[\mathrm{E}(\tau)+\frac{T}{2}\right]+\frac{\lambda^2[\mathrm{E}(\tau^2)+T\mathrm{E}(\tau)+T^2/3]}{2(1-\lambda[\mathrm{E}(\tau)+T/2])}$$

平均待ち時間についてはリトルの公式を使うことができる.

明らかに，FCFS ドラムに対して容量(最大転送速度)は $1/[\mathrm{E}(\tau)+T/2]$ で与えられる．もし，平均すればドラム半回転分に相当する冒頭の待ちを回避できるとするならば，これよりもはるかに良くすることができる．これを行うためには，単にサービス規律を修正し，各意志決定時点で，次にサービスされる要求は，待っている要求の中で開始番地がドラムの現在の角位置の最も近くにあるものになるようにすればよい[2]．この**最短到達時間順**(SATF, shortest-access-time-first)システムの一般的な解析は非常に難かしい[8]．しかしながら，ページング・システムという重要な環境の下では，結果はマルコフ連鎖解析によってむしろ容易に得られる[9]．以下では，そのようなモデルを解析して FCFS 式順序付けと比較する.

システムに到着する要求は1個のページに対するものであって，ドラムの円周上には丁度 $N$ ページ(またはページ枠)が存在すると仮定する．断面図で見るならば，1ページは1ドラム・**セクタ**(sector)を占める.

SATF ページング・ドラムの数学的モデルは図 5.2-1 に示されている．各到来要求は，それの指定する番地を持つセクタに対応するセクタ待ち行列の最後

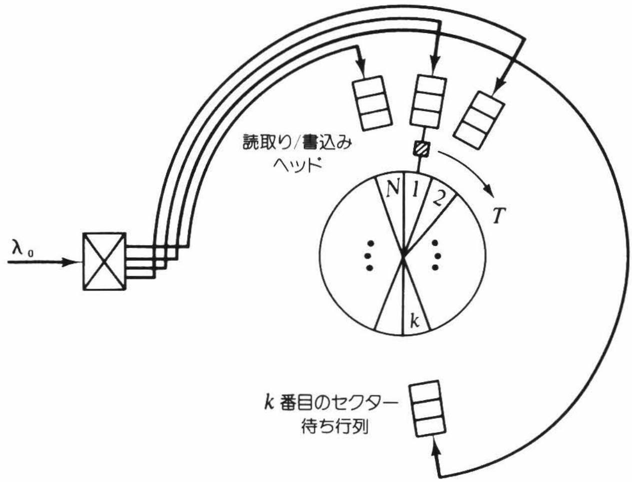



図 5.2-1　数学的モデル

尾に置かれる．ドラムは静止しているとして，読取り/書込みヘッドがドラムの周りを $T$ 秒毎に 1 回転という一定の速さで廻っていると見なすことによって，ドラム・システムの裏返しではあるが，等価な見方をしていることに注意しよう．SATF ページング・ドラムの操作法に従い，読取り/書込みヘッドは，空でない待ち行列を持つセクタ上を通過する度毎に，行列の先頭にある要求に対してサービスを行い，(1 ページが読まれるか書かれるかした後)この要求をシステムから放出する．

到着するページ要求は，定常確率分布 $\{\lambda_k/\lambda_0\}_{k=1}^{N}$ に従って $N$ 個のセクタ待ち行列に分配されるものと仮定する．ただし，$\lambda_0$ はシステムへの平均(総)到着率であり，$\lambda_k (k=1, 2, ..., N)$ は $k$ 番目のセクタ待ち行列への到着率である．そうすると，明らかに次式が成り立つ．

$$\lambda_0 = \sum_{k=1}^{N} \lambda_k \tag{5.2.1}$$

ページ要求はポアソン流れをなして到着率 $\lambda_0$ で到着すると仮定する．したがって，4.2 節で示したように，セクタ待ち行列への到着は，$k$ 番目のそれに対しては到着率が $\lambda_k$ であるような独立なポアソン到着になる．$k$ 番目のセクタに対する到着時間間隔の分布は次式のように表すことができる．

$$(5.2.2) \qquad A_k(t) = 1 - e^{-\lambda_k t}(t \geq 0)$$

各セクタ待ち行列は独立のものとして扱えるにもかかわらず，それらは個々に一定サービス時間の単一サーバを持つ(M/D/1 型)待ち行列であると解釈する訳にはいかないということに特に注意するべきである．さらに言えば，空でない待ち行列への到着から見ると本質的にドラム回転時間 $T$ に等しいサービス時間を持つ一定サービス時間システムではあるが，空である待ち行列への到着は明らかに一定のサービス時間を持たないというのが真実である．

解析の一般的目標は，統計的平衡状態でシステムが作動しているという仮定の下で $N$ 個のセクタ待ち行列の各々の長さの結合分布関数を見出すことである．セクタ待ち行列の統計的独立性から，これは積分布となるであろう．したがって，得る必要のある結果は，単一の待ち行列の長さの確率塊関数(pmf)ということになる．ここでの接近法は，まず，時間を等間隔に分ける時刻の系列を定義して，次に，その結果生じる離散時間における過程に対して，求めたいpmf を算出するというものである．その後で，統計的平衡状態における連続時間過程に対して有効となる結果をこの pmf から決定することにする．離散時間での過程は，ここで仮定した到着機構の結果としてマルコフ性を持つ．

時刻の系列 $\{t_n\}_{n=1}^{\infty}$ は，読取り/書込みヘッドがセクタ $k$ の終端を通過した直後の時刻の系列に対応するように定義される．$N$ 個の待ち行列の中の任意の 1 個(すなわち，$k$ 番目)の行列の長さの定常確率塊関数 $\{\pi^{(k)}{}_i\}_{i=1}^{\infty}$ を見出したい．ここで，$\pi^{(k)}{}_i$ は，特に $k$ 番目の待ち行列の長さが読取り/書込みヘッドのセクタ $k$ の終端通過直後に $i$ となる平衡状態確率である．以下での解析の目的にとっては，$k$ への依存を排除し，$\lambda = \lambda_k$ また $\pi_i = \pi^{(k)}{}_i$ と置いて表記を簡単にしても差支えない．さて，確率 $\pi_i$ に対する下記の母関数の計算を始めよう．

$$(5.2.3) \qquad \pi(z) = \sum_{i=0}^{\infty} \pi_i z^i$$

後に検証するが，モデルから直観的に明らかなように，$k$ 番目の待ち行列における統計的平衡条件は，単に到着率 $\lambda$ が最大退去率 $1/T$ を越えないということである．ところで，統計的平衡の定義によって次式が成り立つ．

$$(5.2.4) \qquad \pi_j = \sum_{i=0}^{\infty} \pi_{ij}\pi_i (j = 0, 1, 2, \ldots)$$

ここで，$\pi_{ij}$ は，系列 $\{t_n\}$ に対して定義される 1 段遷移確率である．すなわ

ち,

(5.2.5) $\pi_{ij}=\mathrm{Pr}[t_n$ において $k$ 番目の待ち行列の長さが $i$ である
という条件の下で, $t_{n+1}$ においてそれは $j$ である$]$

$\pi_{ij}$ を求めるためには, まず, 各期間 $t_{n+1}-t_n=T$ に対して最大限1個の退去しかないので, 全ての $j<i-1$ に対して $\pi_{ij}=0$ であることに注目しなければならない. 同様に, もし $i>0$ ならば,

(5.2.6) $\pi_{ij}=\mathrm{Pr}[j-i+1$ 個の到着が $t_{n+1}-t_n=T$ 内に生じる$]$
$(0\leq i-1\leq j)$

であることも定義から明らかである. $i=0$ に対する $\pi_{ij}$ の計算においては, 読取り/書込みヘッドがセクタ $k$ の終端からセクタ $k$ の始端に戻るまで回転する間の時間$[1-1/N]T$ 内に発生する到着と読取り/書込みヘッドが $k$ 番目のセクタを横切って回転する長さ $T/N$ の時間間隔内に発生する到着とを区別しなければならない. というのは, もし前者の時間間隔内に少なくとも1個の到着があれば, そのような到着の1番目のものは, 読取り/書込みヘッドが $k$ 番目のセクタの終端を再び通過する時刻までにサービスされてしまうからである. したがって, 次式が成り立つ.

(5.2.7) $\pi_{0j}=\sum_{r=1}^{j+1}\mathrm{Pr}\Big[r$ 個の到着が $\left(\dfrac{N-1}{N}\right)T$ 内にあり, $j-r+1$ 個の到着が $T/N$ 内にある$\Big]+\mathrm{Pr}\Big[\left(\dfrac{N-1}{N}\right)T$ 内の到着は 0 個であり, $j$ 個の到着が $T/N$ 内にある$\Big]$

ところで, 第4章で算出したポアソン pmf によれば

(5.2.8)
$$f_j(t)=\frac{(\lambda t)^j}{j!}e^{-\lambda t}$$

である. それゆえ, 式(5.2.6)の右辺は $f_{j-i+1}(T)$ で与えられ, 式(5.2.7)は次のようになる.

(5.2.9)
$$\pi_{0j}=\sum_{r=1}^{j+1}f_r\left(\frac{N-1}{N}T\right)f_{j-r+1}\left(\frac{T}{N}\right)+f_0\left(\frac{N-1}{N}T\right)f_j\left(\frac{T}{N}\right)$$

これは次のように簡単化される.

$$(5.2.10)\qquad \pi_{0j} = f_{j+1}(T) - f_0\Big(\frac{N-1}{N}\,T\Big)\Big(f_{j+1}\Big(\frac{T}{N}\Big) - f_j\Big(\frac{T}{N}\Big)\Big)$$

次に，式(5.2.4)の両辺に $z^j$ を乗じ，$j \geq 0$ に関する総和を取ると，総和の順序を変更し，式(5.2.3)を使った後，次式を得る．

$$(5.2.11)\qquad \pi(z) = \sum_{i=0}^{\infty} \pi_i \sum_{j=0}^{\infty} \pi_{ij} z^j$$

$\pi_{ij}$ の代入を行い多少簡単化すると，次式が得られる．

$$(5.2.12)\qquad \pi(z) = \frac{\pi_0}{z}\Big[A_T(z) - f_0\Big(\frac{N-1}{N}\,T\Big)A_{T/N}(z) + zf_0\Big(\frac{N-1}{N}\,T\Big)A_{T/N}(z)\Big] + \frac{A_T(z)}{z}\,[\pi(z) - \pi_0]$$

ただし，

$$(5.2.13)\qquad A_t(z) = \sum_{j=0}^{\infty} f_j(t) z^j = e^{-\lambda t(1-z)}$$

はポアソン分布に対する母関数である．式(5.2.12)を $\pi(z)$ について解くと，次式が得られる．

$$(5.2.14)\qquad \pi(z) = \pi_0\,\frac{(z-1)e^{-\lambda T(1-z)/N}}{z - e^{-\lambda T(1-z)}}\,e^{-\lambda[(N-1)/N]T}$$

$\pi_0$ を見出すために $\lim_{z\to 1}\pi(z) = 1$ を用いると，次式が得られる．

$$(5.2.15)\qquad \pi_0 = (1-\lambda T)e^{\lambda[(N-1)/N]T}$$

次に，$\pi_i$ だけを基にして，ドラム上の任意の点で定義される pmf が計算できることを述べる．これを証明するため，図 5.2-1 で，セクタ $k$ の終端直後の点から $\tau$ 秒だけずれた点で定義される pmf を見出すことを要求されているものとする．$0 \leq \tau < T$ であるような任意の $\tau$ に対し，$k$ 番目の待ち行列の長さは，読取り/書込みヘッドがセクタ $k$ の終端から $\tau$ 秒だけ回転する間に**ただ増加し得るのみ**である．それゆえ，要求されている pmf は次式で与えられる．

$$(5.2.16)\qquad p_j(\tau) = \sum_{i=0}^{j} \pi_i f_{j-i}(\tau)$$

これは，平衡状態における $k$ 番目の待ち行列の状態のある観測に対し，もし

その観測時点を示す $\tau$ が与えられれば，式(5.2.16)から対応する pmf が知られるということを意味する．したがって，連続時間におけるこの過程の行列長の確率 $p_j$ を得るには，単に，区間[0, $T$)における $\tau$ の条件付けを排除すればよい．すなわち，次式を得る．

$$p_j = \int_0^T p_j(x)\,dF(x) \tag{5.2.17}$$

ただし，$F(x)$ は $\tau$ の分布関数を表す．明らかに，$\tau$ は区間[0, $T$)において一様に分布するので，次の式が成り立つ．

$$dF(x) = \frac{dx}{T} \quad (0 \leq x < T) \tag{5.2.18}$$

前と同様に母関数を用いて議論を進めると，$p_j(\tau)$ の母関数が次のように算出される．

$$p(z, \tau) = \pi(z) A_\tau(z) \tag{5.2.19}$$

ここでは，式(5.2.16)が畳込みを表しているという事実を利用している．式(5.2.17)を変形し，式(5.2.13)[†] を代入すると，次式を得る．

$$p(z) = \sum_{j=0}^{\infty} p_j z^j = \pi(z) \int_0^T A_x(z)\,dF(x) \tag{5.2.20}$$

式(5.2.13)～(5.2.15)および(5.2.18)[††] を代入して積分を実行すると，次の式が得られる．

$$p(z) = (1-\lambda T)\,\frac{(z-1)e^{-\lambda T(1-z)/N}}{z - e^{-\lambda T(1-z)}}\left[\frac{1-e^{-\lambda T(1-z)}}{\lambda T(1-z)}\right] \tag{5.2.21}$$

式(5.2.21)は，求めようとしていた基本的な結果である．式(5.2.21)からは，確率 $p_j$，平均行列長 $\overline{n}$，および高次のモーメントが微分を行うことによって求められる．特に，平均行列長については次式のようになる．

$$\overline{n} = \lim_{z \to 1} P'(z) = \rho\left(\frac{N+2}{2N}\right) + \frac{\rho^2}{2(1-\rho)} \tag{5.2.22}$$

---

†訳注　原文は式(5.2.18)となっているが，本文のように訂正した．
††訳注　原文にはないが，(5.2.18)を追加した．

ただし，$\rho$ は($k$ 番目の待ち行列の)利用率であって，次式で与えられる．

$$\rho = \lambda T \tag{5.2.23}$$

システムに対する平均待ち時間の尺度はリトルの公式(4.2 節を見よ)を適用することによって与えられる．したがって，システムにおける待ち時間については，単に $W = \overline{n}/\lambda$ を計算するだけでよい．

平均行列長に対する式(5.2.22)からセクタの個数($N$)への依存性が看て取れる．実際のシステムにおいて最も興味ある例には $\lambda \equiv \lambda_k = \lambda_0/N$ という仮定が含まれている．大概のページング・ドラムは $(N+2)/2N$ が十分に 1/2 に近くなるくらい大きい $N$ の値を使用しているので，システム性能は本質的には**負荷係数**(loading factor) $\rho = \lambda T = \lambda_0 T/N$ に依存することになる．所与のドラム速度$(1/T)$ とページの大きさに対して数 $N$ はドラムの大きさ(円周)の尺度となる．ドラムの大きさと速度を固定すると，$N$ はページの大きさの尺度となる．いずれにしてもシステムの**容量**(飽和点)は $N$ の値に直接比例する．

先に示唆したように，ページング・ドラムに対して与えた解析は，各セクタ待ち行列が一定時間サービスの単一サーバ・システムであると仮定することによって近似することができる．4.2 節で導いた結果[式(4.2.24)を見よ]を使用すると，$k$ 番目の待ち行列の平均の長さに対して次式を得る．

$$\overline{n} = \rho + \frac{\rho^2}{2(1-\rho)} \tag{5.2.24}$$

これは式(5.2.22)と比較される．$N=2$ に対しては式(5.2.22)と(5.2.24)は同じものになり，式(5.2.24)は $N=1$ を除く全ての $N$ に対して控え目の評価となる．$N=1$ に縮退した場合については，現実のシステムでの空な待ち行列への到着はサービス完了までに平均して $3T/2$ 秒待つことになるが，上記の近似によれば $T$ 秒だけの待ちになるということが看て取れよう．最後に，式(5.2.22)と(5.2.24)の間の相違の割合は，$\rho$ が大きくなると非常に小さくなるということも視察することができる．すなわち，項 $\rho^2/2(1-\rho)$ が両システムにおいて $\overline{n}$ の値を支配している．

1ページのみの要求がポアソン到着するという仮定の下では，ページング・ドラムに対する上述の結果は，完全な SATF スケジューリングのタイミングと制御に関する要請に適合し得ない実在のシステムがこれよりも貧弱な性能と低い容量で特徴付けられるであろうという意味で，性能の上限を与えるものであると見ることができよう．下限は，サービス待ちのページ要求がどのセクタ

番地を指定しているかを知っていることの利点を用いない FCFS スケジューリングのシステムによって与えられる．

FCFS ページング・ドラムに対する妥当な近似を得るため，全ての $k$ に対して $\lambda_k = \lambda_0/N$ を仮定し，単純に $M/G/1$ に対する式(4.2.21)の結果を用いる．その際，サービス時間は，等確率$(1/N)$で $mT/N(m = 0, 1, \ldots, N-1)$ の中のどれかが値として取られるような到達時間と定数 $T/N$ との和から成るものと仮定する．その結果，次式が得られる．

$$\overline{n}_0 = \rho_0 + \frac{\lambda_0 \mathrm{E}(S^2)}{2(1-\rho_0)} \quad \left(\rho_0 = \frac{\lambda_0}{\mu}\right) \tag{5.2.25}$$

ただし，$\mathrm{E}(S^2)$ はサービス時間分布の 2 次モーメント，すなわち，

$$\mathrm{E}(S^2) = \left[2(N+1)\,\frac{2N+1}{3}\right]\left(\frac{T}{2N}\right)^2$$

であり，$1/\mu$ は平均値，すなわち，

$$\frac{1}{\mu} = (N+1)\,\frac{T}{2N} \tag{5.2.26}$$

である．

FCFS システムをよく検討すれば，たとえ，空である待ち行列への到着に対してはサービス時間分布に違いがあるために齟齬が存在するにしても，上の結果は精密なものであることがはっきりするはずである．この齟齬は，$N$ か負荷かが増大すると共に減少する．なぜならば，前者の場合には $T/N$ は無視できる程のものになり，後者の場合には待ち行列が空である確率は零に近付くからである．SATF 規律と FCFS 規律の間に存在する大きな相違についての観念を得るために行わなければならないことは，実際の所，それらの容量を互いに比較することだけである．これについては，式(5.2.23)と(5.2.26)から†，与えられた $N$ に対して，最短到達時間順システムは FCFS システムよりも容量が $(N+1)/2$ 倍大きいということが容易に知られる．

ハードウェアによる遅れあるいは処理における遅れがあるため，引続くセクタに対して連続サービスを行うことが妨げられるような場合があるが，このような場合に SATF スケジューリングを実現する一つの方法は，$1<M<N$ で

†訳注　原文は式(5.2.22)と(5.2.25)になっている．

あるようなある適当な整数 $M$ を定め，ドラムの回転の進行に伴って $M-1$ 個置きに現われるセクタに対してだけサービスするという方式を採用することである．**歳差ドラム方式**(precessing drum scheme)は今述べたような特徴を持っている[10]．なお，先に導いた結果は，わずかな修正を加えるだけでこの方式にも適用することができる．

歳差ドラム方式における上記の $M$ は，$1<M<N$ であって，しかも $M$ と $N$ は互いに素であるという条件を満たす任意の整数である．(後者の条件が満足されない場合にも有効であるように方式を調整することは比較的に簡単である．)この方式の動作を説明するため，簡単な例として，$M$ で $N-1$ が割切れるような場合(たとえば，$M=5$，$N=26$)を採上げよう．セクタ $M$ から開始して $M$ セクタ毎にサービスを行うある回転が存在すると仮定しよう．セクタ $N-1$ のサービスの後，次に($M-1$ 個のセクタを飛び越した後で)サービスされることになるセクタはセクタ $M-1$ である．このように，後に続く回転でサービスされるセクタは，1回前の回転でサービスされたセクタから1だけ前にずれることになる．こうして，歳差を $M$ 回転続けた後，引続いて，セクタ $M$ から再度同じことが繰返される．このように操作することによって，各セクタは $M$ 回転毎に丁度1回のサービスを受け，しかも引続いてサービスされる二つの待ち行列の間には飛び越されるセクタが丁度 $M-1$ 個存在するということになる．サービスの対象として次から次へと現われるセクタの系列は $M$ に依存して定まるけれども，最後に述べた二つのことは，前記の条件を満たすどのような $M$ に対しても成立する．

ところで，もしも新しい到着が，それがサービスされ得るようになる前に少なくとも $(M-1)T/N$ 秒を必要とすると仮定するならば，前述の解析は，前に導いた全ての結果において単に $T$ を $MT$ で置換えることにより，ただちに有効となる．このことは，円周が元のドラムのそれの $M$ 倍であるにもかかわらず，読取り/書込みヘッドを過ぎる円周の速さは変わらないような等価なドラムを考えることにより最も容易に理解される．この等価ドラムのセクタは元のドラム上の $M$ 個のセクタ(サブセクタと呼ばれる)から成る．しかし，実際に情報を含むのはこの $M$ 個のサブセクタの中のただ1個だけである．もし，情報を含むのは $M$ 個のサブセクタの中の最後尾のものであるというように等価ドラムのセクタを定義するならば，$T$ を $MT$ で置換えることにより，$P(z)$ を導く前述の解析は何も変更することなくそのまま通用する．

Burge と Konheim[11]はシステム内の要求総数(サービス中の1個を含む)が与えられた一定の大きさ $L$ に保たれるという仮定の下で，SATF ページン

グ・ドラムの解析を行った．すなわち，ある要求が退去するとき，かつそのときに限り，新しい要求が編入されて，$1/N(1\leq k\leq N)$の確率で$k$番目の待ち行列に加えられるというものである．このシステムのマルコフ連鎖解析が詳細に展開されているにもかかわらず，その主な工学的結果はリトルの公式の応用によって得ることができる．

$w$は，$k$番目の待ち行列への編入を許される要求がシステム内で費やす時間を表す確率変数であるとする．対称性により，これは全ての$k$に対して同じになる．下げ札付きの要求と呼ばれる与えられた要求に対して次のように書くことができる．

$$\Pr[w\leq x] = \Pr[w' + w'' + w'''\leq x]$$

ただし，$w'$は編入許可の瞬間からヘッドが回転してセクタ$k$の始端に達するまでの経過時間，$w''$はそれに続いて到着時にシステム内にあるセクタ$k$への要求の全てをサービスするのに要する時間間隔，また，$w''' = T/N$は単に下げ札付きの要求を処理するための時間である．期待値を取ると次のようになる．

$$W\equiv \mathrm{E}(w) = \mathrm{E}(w') + \mathrm{E}(w'') + \frac{T}{N}$$

さて，下げ札付きの要求はある退去の直後に編入を許され，また，この退去はどのセクタからでも確率的に同等に生じ得るから，$\mathrm{E}(w') = [(N-1)/2](T/N)$となる．次に，$\mathrm{E}(w'') = \overline{n}_k T$と書ける．ただし，$\overline{n}_k$は下げ札付きの要求の前にあるセクタ$k$への要求の平均個数である．対称性から，また下げ札付きの要求は到着時に全部で丁度$L-1$個の要求をシステム内に見出すはずであるから，$\overline{n}_k = (L-1)/N$となる．こうして，

$$W = \frac{N-1}{2}\frac{T}{N} + (L-1)\frac{T}{N} + \frac{T}{N}$$

すなわち

(5.2.27)
$$W = \frac{2L+N-1}{2}\frac{T}{N}$$

がシステム内で費やす時間の平均となる．リトルの公式から

(5.2.28)
$$L = \lambda W$$

となる．ただし，$\lambda$は要求がシステムに入るのを許す率である．それゆえ，ド

ラムの回転当りサービスされる要求数の平均は次式で与えられる.

$$\rho_L = \lambda T = \frac{LT}{W} = \frac{2\,NL}{2\,L + N - 1}$$

予期した通り, $\lim_{L\to\infty}\rho_L = N$ である. これは先に解析した SATF ページング・ドラムに対する飽和時の結果であり, $\rho_1 = 2\,N/(N+1)$ は, 式(5.2.26)に従う FCFS システムに対する飽和時の結果 $\mu T$ に等しい.

## 5.3 探索時間の影響の最小化

トラックからトラックへとヘッドを動かさなければならないディスク・システムにおいて生じる第一のサービス段階の問題に取掛かることにしよう. 一般には, ヘッド位置決めの問題と回転待ち時間の問題は分離して研究することができる. というのは, 興味ある大抵の事例では二つのサービス段階を結合して最適化を図ることに何の利点もないように見えるからである [6]. 前に示唆したように, ドラム・システムに対して解析した SATF 方式は, ディスク回転における回転待ち時間の影響の最小化に対しても同じように具合よく適用することができる. しかしながら, 多くのディスク・システムにおいては, 探索時間が非常に大きいので, 回転待ち時間を最小にすることは全体の性能にほとんど影響を持たないという事実は注目するに価する. 当然, このことは, ディスク・システムにおけるヘッドの動きを減少させるため以下で提供し解析する技法の重要性を増大させる. 特にディスク入/出力が総合システム性能を左右する要因になっているときには, なおさらそうである.

本節では今後, 関心の対象を探索段階でのヘッド(アーム)の位置決め操作に限定することにする. さらに, 特定のシステムの細部にわたる解析においては一般にヘッド切り換え時間を考慮に入れなければならない [2] であろうけれども, ここでは, ディスクの読取り操作と書込み操作の間にある違いには全くかかずらわないことにする. 本節では, 要求の待ち行列に対応して行う必要がある探索に対する様々なスケジューリング技法の数学的モデルの比較研究に注意を集中する. 最初の議論を簡単化するため, **図 5.3–1** に示すような単純なモデルを採用しよう. そこでは, 1 個のヘッドと $N$ 本のトラックを持つ 1 面のディスクが示されている.

いつものように, 最初に考える規律は FIFO 式順序付けである. 特に, ディスク面に情報を読み書きする要求は 1 本の FIFO 待ち行列内に蓄えられると仮定する. すなわち, 各意志決定時点で行列の先頭にある要求が選択される

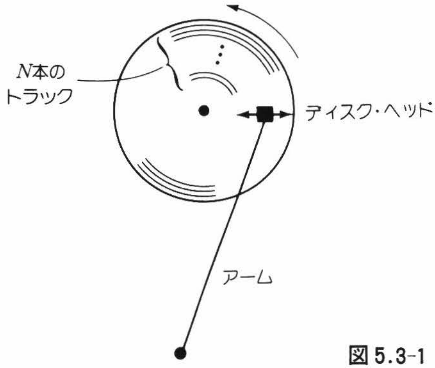



**図 5.3-1**　1面のディスク

と，ディスク・ヘッドは(もし必要ならば)その要求に適合するシリンダに向かって移動を始める．このシステムに関する待ち行列解析は，第4章の結果の決まりきった応用に帰するので，後は読者に任せせることにする．

ここで，FIFO 規律におけるヘッドの位置決めに起因する時間遅れの期待値にまず注目しておくことにしよう．$N$ 本のトラックがあるものとし，引き続く要求のトラック番地は集合 $\{1, 2, \ldots, N\}$ から互いに独立にかつランダムに選ばれるものと仮定する．一つの要求にサービスを施す際にヘッドが上を横切って移動しなければならない平均のトラック数は漸近的に $N/3$ となるということは容易に示される．ドラム・スケジューリングの場合と同様に，待ち行列中の諸要求のトラック番地は分っていると仮定されるときは，その情報の利用によって冒頭の時間遅れを著しく減少させることができる．

SATF 式順序付けへの拡張として，サービス向上の一つの可能な方法はいわゆる**最短探索時間順**(shortest-seek-time-first, SSTF)手続きである [2]．SSTF 式順序付けを行う各意志決定時点でサービスのために選ばれる次の要求はヘッドの現在位置に最も近いトラック番地を持つものである．少々考えれば，SSTF 規則の一つの明白な欠点が露になる．すなわち，ディスク面の内側端と外側端にある(高い番号と低い番号の付いた)トラックは内部のトラックに比較して貧弱なサービスしか期待することができない．しかしながら，FIFO 式順序付けと比較すると，(低負荷時における)上記の差別待遇は，大変具合のよいことに，システム容量の著しい増大の代償として支払わねばならない対価としては僅かなものである．

(全てのトラックが一様に使用されるということはここでも仮定する．) SSTF 規則のシステム容量を保持しつつ，側端のトラックに対する差別待遇を僅

少なものにするために有効な二つの方法がある．第一の規則は，SCAN 規則 [2] と呼ばれ，単純に1方向に対してのみ SSTF 規準を適用するものである．すなわち，ヘッドは与えられた方向にトラックを走査し，要求に出会い次第それに対するサービスを行う．この操作は，下記の二つの事象の中の一つが生じるまで続けられる．

1. 与えられた方向の最後のトラックが処理されてしまう．または，
2. 与えられた方向ではヘッド位置の前方にもはや要求は存在しない．

いずれにしても，ヘッドの動きは逆向きにされ，走査は，(待ち行列が空でないと仮定すると)新たな方向に向かって続行され，上記の事象の中の一つが再び生じるまで実施される．

第二の方法は，FSCAN と名付ける FIFO 式走査規則である [5]．各意志決定時点で FSCAN は，待ち行列内の全要求を取って来て1回の走査でサービスしてしまう．走査の方向は，サービスされる要求の中での最外側および最内側のトラック番地と現在のヘッド位置を比較してより近い方から始められるように決められる．それゆえ，走査に先立って側端のトラック番地の近い方(すなわち，最小の番地と最大の番地のうち近い方)へ向かってディスク・ヘッドの移動が行われる．所与の走査の途中で発生する到着は，待ち行列内に留め置かれ，次の走査までサービスされない†．これが SCAN 規則と FSCAN 規則の間の主な相違点である．中程度以上の負荷が掛かっている期間中は，SCAN 規則と FSCAN 規則のどちらを用いても，ヘッドはディスク面を横切って左右に広く振られるであろうことは明らかである．今，左から右へ向う走査を想定するとしよう．番地は左から右へ向って大きくなるように定められていると仮定すると，FSCAN 規則に従って一群の要求がサービスされた直後には，ヘッドは恐らく高い番地の所にあろう．また，この走査の間に，多分，数個の要求が到着しているに違いないと考えると，待ち行列中の要求が持っているトラック番地の中で最大のものが現在のヘッド位置に最も近いものであろうから，次には右から左への走査が惹起されることになろうということがいえる．

さて，SCAN 式順序付けと FSCAN 式順序付けの間の構造的な類似と相違を特徴付けることに取掛かろう．SCAN による応答性は FSCAN によるそれよりも(全てのトラック番地の上で)一様によいけれども，SCAN には依然として最内側と最外側のトラックに対するある程度の冷遇が含まれるということが以下で示される．

---

†訳注 FSCAN は FIFO 式であると述べた前文のいわれがこの条件である．

数学的モデルは次のように記述される．離散的トラック番地の集合を，区間 $[0, d]$，すなわち，$[0, d]$ 内の実数の集合で置換える．簡単のため，しかし一般性を失わずに，$d = 1$ と仮定し，ヘッドがこの区間を横切って移動するのに要する時間は $a$ であると仮定する．到着率 $\lambda$ のポアソン入力を仮定し，トラック番地は区間 $[0, 1]$ の上で一様に分布するものとする．形式的には，ある与えられた到着が区間 $(x, x + \Delta x)$ 内に落ちる確率は，$\Delta x/d = \Delta x$ で与えられる．この仮定は，与えられたどの時刻にもトラック当りに要求が高々1個存在するという単純化に矛盾しない．非形式的には，微小な区間 $\Delta x$ 内に1個より多くの到着がある確率は $(\Delta x)^2$ の位数であるから，区間 $\Delta x$ がトラックに対応しているものと見なしてよい．明らかに，この近似は，実際のトラック数 $N$ が大になるに従って向上する．

どの要求をサービスするにしても，それに要する時間は一定値 $T$ であると仮定しよう．ここで仮定をもっと一般的に設定することもできるけれども，それは行おうとしている形での比較にとって本質的ではない．ここでは，明らかに，回転待ち時間と転送時間を $T$ に一括しているだけでなく，ヘッド移動の開始と停止によって惹起される遅れとヘッド切換え時間も一纏めにして $T$ に含めている．最後に，**ヘッドは，それが要求にサービスを施していないときには常に走査中であり，しかも，走査の方向は，ヘッドが番地0または1において境界に到達したときにだけ**† **逆向きにされる**と仮定することにする．SCAN 規則の解析における仮定は次の通りである．

1．もしヘッドの1度の横断において $n$ 個の要求がサービスされるならば，横断時間は $nT + a$ で与えられる．
2．区間 $(b, b'), 0 \leq b < b' \leq 1$ へ時間 $t$ 内に到着する個数 $(n)$ の pmf は，下記のポアソン分布で与えられる．

$$f_n(b, b') = \frac{[\lambda(b' - b)t]^n}{n!} e^{-\lambda(b'-b)t} \tag{5.3.1}$$

平均値は次の通りである．

$$\bar{f}_n(b, b') = \lambda(b' - b)t \tag{5.3.2}$$

上述の諸仮定の中の一部の緩和は，後に，また，演習問題の中でも考えられる．

† 一般モデルのこの簡単化は，SCAN 規則の解析にとっては本質的であるが，FSCAN 規則にとってはそうでないように見える．

SCAN 規則の解析から始めよう．目標は，第一に，極限の統計的平衡状態において0から $x$ までヘッドが動くのに要する平均時間の表現式を求めることであり，第二に，平均待ち時間または平均応答時間の一つの尺度を得ることである．

便宜上，ディスク・ヘッドの奇数番目の横断は左から右へ向うものと仮定しよう．$y_{2n+1}(x)\ (n\geq 0)$ は，ヘッドが $(2n+1)$ 番目に左から右へと横断する途上で出会う要求をサービスしながら位置0から距離 $x$ だけ移動するのに要する時間を値とする確率変数を表すものとする．同様に，$(2n)$ 番目の横断に対しては，$y_{2n}(x)$ が，位置1から左へ向って距離 $x$ の移動をするのに要する時間に対応する確率変数を表す．

SCAN 規則の記述から(図5.3-2を見よ)，$(2n+1)$ 番目の横断に先立って，区間 $(x, x+\Delta x)$ がサービスを受ける機会を最後に持ったのは，ヘッドが $(2n)$ 番目の横断中にこの区間を通って右から左へ動いていた間であったということが分る．それゆえ，$y_{2n+1}(x)$，$y_{2n}(1-x)$，および $y_{2n}(1)$ が与えられれば，式(5.3.2)から，$(2n+1)$ 番目の横断の途中において区間 $(x, x+\Delta x)$ 内でサービスが行われる要求の平均個数は，$\lambda\Delta x\,[y_{2n+1}(x)+y_{2n}(1)-y_{2n}(1-x)]$ と無視することが可能な $(\Delta x)^2$ の位数の項との和になる．そのような要求の各々は $T$ 時間単位を必要とし，またヘッドが区間を通過するのに $a\Delta x$ 時間単位を要するから，次式が成立する．

(5.3.3)
$$\begin{aligned}&\mathrm{E}\,[y_{2n+1}(x+\Delta x)\,|\,y_{2n+1}(x),\ y_{2n}(1),\ y_{2n}(1-x)]\\&\quad=y_{2n+1}(x)+a\Delta x+\lambda T\Delta x\,[y_{2n+1}(x)+y_{2n}(1)-y_{2n}(1-x)]\end{aligned}$$

$\mathrm{E}\,[y_i(x)]\equiv\bar{y}_i(x)$ と定義し，式(5.3.3)における条件付けを取り除くと，次式を得る．

(5.3.4)
$$\begin{aligned}\bar{y}_{2n+1}(x+\Delta x)=\bar{y}_{2n+1}(x)+a\Delta x+\lambda T\Delta x\,[&\bar{y}_{2n+1}(x)\\&+\bar{y}_{2n}(1)-\bar{y}_{2n}(1-x)]\end{aligned}$$

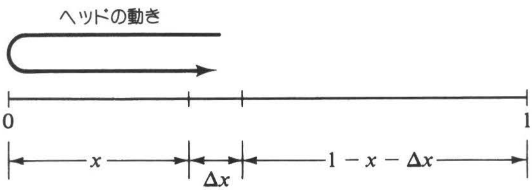



図5.3-2　$y(x)$ の決定

さて，極限 $\bar{y}(x) \equiv \lim_{n\to\infty}\bar{y}_{2n+1}(x)$ を考えよう．この問題に対する仮定の下では，極限の存在は条件 $\lambda<1/T$ で保証される．言い換えれば，統計的平衡が存在するためには，最大サービス率 $1/T$ が到着率 $\lambda$ より大であることを必要とする．さらに，問題の対称性から，$\lim_{n\to\infty}\bar{y}_{2n}(x)=\bar{y}(x)$ であることが容易に頷ける．こうして，式(5.3.4)において極限を取ると，次式のようになる．

$$\bar{y}(x+\Delta x)=\bar{y}(x)+a\Delta x+\lambda T\Delta x[\bar{y}(x)+\bar{y}(1)-\bar{y}(1-x)]$$

$\Delta x$ で割り，極限 $\Delta x\to 0$ を取ると，次式が得られる．

$$\frac{d\bar{y}(x)}{dx}=a+\lambda T[\bar{y}(x)+\bar{y}(1)-\bar{y}(1-x)] \tag{5.3.5}$$

式(5.3.5)を解くため，$d\bar{y}(x)/dx\equiv\bar{y}'(x)$ と置く．式(5.3.5)の結果，次の関係式が成り立っていることに着目する．

$$\bar{y}'(x)+\bar{y}'(1-x)=2\,\bar{y}'\left(\frac{1}{2}\right)$$

式(5.3.5)を微分したものにこの関係式を用いると，次式が得られる．

$$\frac{d^2\bar{y}(x)}{dx^2}=2\,\lambda T\frac{d\bar{y}(x)}{dx}\bigg|_{x=\frac{1}{2}}=\text{定数} \tag{5.3.6}$$

この式から $\bar{y}(x)$ が $x$ の2次式でなければならないことが分る．
式(5.3.5)以外に境界条件 $\bar{y}(0)=0$ と $d\bar{y}(x)/dx|_{x=0}=a$ を用いると，次の結果を得る．

$$\bar{y}(x)=\frac{1-\lambda T(1-x)}{1-\lambda T}ax \quad (0\leq x\leq 1) \tag{5.3.7}$$

$\bar{y}(1)$，すなわち，平均横断時間は次式で与えられることに注意しよう．

$$\bar{y}(1)=\frac{a}{1-\lambda T} \tag{5.3.8}$$

式(5.3.8)は，次の独立した議論によって確かめることができる．平均横断時間が次のように書けることに着目する．

$$\bar{y}(1)=\bar{n}_cT+a \tag{5.3.9}$$

ただし，$\bar{n}_c$ は1回の横断でサービスされる個数の(平衡状態における)平均である．しかし，$\bar{y}(1)$ 内にサービスされる平均個数は $\bar{y}(1)$ 内に到着する平均個数に他ならない．したがって，

$$\overline{n}_c = \lambda \bar{y}(1) \tag{5.3.10}$$

式(5.3.9)に代入すると，求める式(5.3.8)が得られる．

さて，統計的平衡状態にあるときに，ヘッドが位置 $v$ にあって右に移動中であるシステムを観察するものとしよう．$w_r(x|v)$ は,ヘッドが現在位置 $v$ から位置 $x$ へ動くのに要する時間の期待値を表すとする．このとき，次式が成り立つ．

$$w_r(x|v) = \begin{cases} \bar{y}(x) - \bar{y}(v) & (0 \leq v \leq x \leq 1) \\ \bar{y}(1) - \bar{y}(v) + \bar{y}(1-x) & (0 \leq x < v \leq 1) \end{cases} \tag{5.3.11}$$

次式

$$\bar{w}_r(x) = \int_0^1 w_r(x|v)\, dP_r(v) \tag{5.3.12}$$

を計算するには，ディスク・ヘッドが右へ移動中であるという条件の下で区間 $(0, v)$ 内にある定常確率分布 $P_r(v)$ が必要である．$P_r(v)$ は，統計的平衡状態においてヘッドが右への移動に費やす時間の内，区間$(0, v)$内で費やす部分の割合の期待値に等しいと見られる．ところで，$M$ が大きい数であるとすると，$M$ 回の左から右への横断に要する時間は近似的に $M\bar{y}(1)$ である．この時間の内，ヘッドは近似的に $\bar{y}(x)M$ 時間単位を区間$(0, x)$内で費やす．漸近的に，次式が成り立つ†．

$$P_r(v) = \frac{\bar{y}(v)}{\bar{y}(1)} \tag{5.3.13}$$

対応する密度関数は次の通りである．

$$p_r(v) = \frac{d\bar{y}(v)/dv}{\bar{y}(1)} \tag{5.3.14}$$

式(5.3.11)の表現を式(5.3.12)に代入すると，次のようになる．

$$\bar{w}_r(x) = \int_0^x [\bar{y}(x) - \bar{y}(v)]\, p_r(v)\, dv + \int_x^1 [\bar{y}(1) - \bar{y}(v) + \bar{y}(1-x)]\, p_r(v)\, dv$$

ここで式(5.3.14)を代入し積分を実行すると，次式が求まる．

† これは，統計的平衡状態においてある与えられた事象の確率分布を決定する際に通常用いられる論法である．これは後にも用いる機会がある．しかしながら，一般に，扱っている確率過程はこの論法によって妥当性と意味ある結果がもたらされるようなものであることが保証されるように注意しなければならない．この型の問題とエルゴード性についての一般問題の詳細な取扱いは本書の範囲を越えている．読者は文献[12, 13]を参照されたい．

(5.3.15) $$\bar{w}_r(x) = \frac{a/2}{1-\lambda T}[1+2(1-x)(1+\lambda Tx)(1-2x)]$$

$\bar{w}_r(x)$ の計算においてはディスク・ヘッドの左から右への横断を仮定した．しかし，もしヘッドが左へ向かって動いていたと仮定するならば，$\bar{w}_l(x) = \bar{w}_r(1-x)$が位置 $x$ へのヘッドの移動に要する時間の平均となる．ランダムな観察によってヘッドの右への動きが見出される確率は，ヘッドの左への動きを見出す確率に等しく $\frac{1}{2}$ である．（しかし，ヘッドの位置が $z \neq \frac{1}{2}$ にあるという条件を付けたときは，ヘッドの右向きと左向きの動きの確率は等しくないということに注意せよ．）それゆえ，ヘッドの動作方向についての条件付けを除くと，総合応答時間として次式が成り立つ．

$$\bar{w}(x) = \frac{1}{2}(\bar{w}_r(x) + \bar{w}_l(x))$$

代入の結果これは次のようになる．

(5.3.16) $$\bar{w}(x) = \frac{a}{2}\frac{1+(1-2x)^2}{1-\lambda T}$$

これが求めていた結果である．この式は外側端と内側端にあるトラックに対する冷遇を明瞭に示している．一見して分るように，$\bar{w}(x)$ は，$x = \frac{1}{2}$ で最小になり，$x = 0$ および $x = 1$ で最大となる．

さて次に，ヘッドの走査方向が反転されるのは，ヘッドがディスク面の側端まで走査し終ったときに限ると仮定される理想化 FSCAN 規則に対して，前述の式に対比される応答時間を計算することにする．平衡状態におけるランダムな時点で，しかも進行中の横断は $n$ 個の要求をサービスしつつあるという条件の下で，FSCAN システムを観察するものと仮定しよう．$\bar{w}_n(x)$ は，ヘッドの現在の横断中に到着し，かつ，次の横断でヘッドがトラック $x$ に到る以前にサービスされることになる全ての要求をサービスした後に，位置 $x$ へヘッドが到達する時点まで待たなければならない観察時点からの時間の平均を表すものとする(そのような要求は，もしヘッドが現在左へ向かって動いているならば[0, $x$)内のトラックに対するものであるし，ヘッドが現在右へ向かって動いているならば($x$, 1]内のトラックに対するものである)．すなわち，$\bar{w}_n(x)$ は，1)現在の横断を完了するための平均時間と 2)現在の横断中に発生し，次の横断において位置 $x$ へヘッドが到達する前にサービスされることになる全ての到着を処理するための平均時間との和となる．ヘッドの移動方向は左右とも等

しい確率で起こるから，また現在の横断は正確に $t_n = nT + a$ の長さでなければならないから，次式が成り立つ.

$$\bar{w}_n(x) = \frac{t_n}{2} + \frac{1}{2}[\lambda T t_n x + ax] + \frac{1}{2}[\lambda T t_n(1-x) + a(1-x)]$$

すなわち

(5.3.17) $$\bar{w}_n(x) = \frac{t_n}{2}[1 + \lambda T] + \frac{a}{2}$$

この式は $x$ に無関係であるということに注意しよう．当面の目標は次式で表される平均応答時間を計算することである.

(5.3.18) $$\bar{w}(x) = \sum_{n=0}^{\infty} p_n \bar{w}_n(x)$$

ただし，$p_n$ は，ランダムな観察によってシステムが $n$ 個の要求をサービスする横断を遂行中であることが見出される定常確率である．$p_n$ を計算するため，まず，$k$ 番目の横断中にサービスされる要求の個数が $X_k$ であるマルコフ連鎖 $\{X_k\}$ の定常状態における振舞を解析する．$\{q_j\}$ はその定常確率分布を表し，また，

$$q_{ij} = \Pr[X_{k+1} = j | X_k = i]$$

はその1段遷移確率を表すものとする．$\{X_k\}$ が一様(homogeneous)であること，したがって，$q_{ij}$ が時間パラメータ $k$ に依存しないことは容易に確かめられる．さらに，全ての $i \geq 0$ および $j \geq 0$ に対して，$q_{ij}$ は，単に $t_i = iT + a$ 時間内に $j$ 個のポアソン到着が生じる確率であるに過ぎない．それゆえ，次式が成り立つ.

(5.3.19) $$q_{ij} = \frac{(\lambda t_i)^j}{j!} e^{-\lambda t_i}$$

定常分布を定義する式

(5.3.20) $$q_j = \sum_{i=0}^{\infty} q_{ij} q_i \quad (j = 0, 1, 2, \ldots)$$

および母関数

(5.3.21) $$Q(z) = \sum_{j=0}^{\infty} q_j z^j$$

から次式が得られる.

$$Q(z) = \sum_{i=0}^{\infty} q_i \sum_{j=0}^{\infty} q_{ij} z^j$$

式(5.3.19)を代入し，簡単化すると，次式が得られる.

$$Q(z) = \sum_{i=0}^{\infty} q_i e^{-\lambda t_i (1-z)}$$

$t_i = iT + a$ を代入すると，次式に到る†.

(5.3.22)　$$Q(z) = e^{-\lambda a(1-z)} Q(e^{-\lambda T(1-z)})$$

式(5.3.22)から，微分することによりモーメントを見出すことができる．特に，最初の二つのモーメントは次の通りである††.

(5.3.23)　$$\overline{n}_c = \frac{\lambda a}{1 - \lambda T}$$

(5.3.24)　$$\overline{n^2{}_c} = \frac{(\lambda a)^2 + 2\,\lambda^2 a T \overline{n}_c + \overline{n}_c}{1 - (\lambda T)^2}$$

式(5.3.23)は式(5.3.10)に対応するものであることに注意しよう．この式は，$\bar{y}(1)$ に対して式(5.3.8)で与えた値を裏付けるために行ったのと同じ論法を用いて予想するべきであった．式(5.3.13)を正当化するために用いた議論を援用

---

†訳注

$$\begin{aligned} Q(z) &= \sum_{i=0}^{\infty} q_i e^{-\lambda t_i (1-z)} \\ &= \sum_{i=0}^{\infty} q_i e^{-\lambda (iT+a)(1-z)} \\ &= e^{-\lambda a(1-z)} \sum_{i=0}^{\infty} q_i \left(e^{-\lambda T(1-z)}\right)^i \\ &= e^{-\lambda a(1-z)} Q\left(e^{-\lambda T(1-z)}\right) \end{aligned}$$

††訳注　$Q(z)$ の1階の導関数は次の通り求められる.

(A)　$$\begin{aligned} \frac{dQ(z)}{dz} &= \lambda a e^{-\lambda a(1-z)} Q\left(e^{-\lambda T(1-z)}\right) + \lambda T e^{-\lambda(a+T)(1-z)} \frac{dQ(w)}{dw} \\ &= \lambda a Q(z) + \lambda T e^{-\lambda(a+T)(1-z)} \frac{dQ(w)}{dw} \quad (w = e^{-\lambda T(1-z)}) \end{aligned}$$

ところで，$z \to 1$ ならば，$w \to 1$ であるから，

$$\begin{aligned} \lim_{z\to 1} \frac{dQ(z)}{dz} &= \lim_{z\to 1} \lambda a Q(z) + \lim_{z\to 1} \lambda T e^{-\lambda(a+T)(1-z)} \lim_{w\to 1} \frac{dQ(w)}{dw} \\ &= \lambda a + \lambda T \lim_{w\to 1} \frac{dQ(w)}{dw} \end{aligned}$$

したがって，

$$\overline{n_c} = \lim_{z\to 1} \frac{dQ(z)}{dz} = \frac{\lambda a}{1 - \lambda T} \quad \nearrow$$

して，$p_n$ と $q_n$ の間の次の関係式を立てることができる．

$$p_n = \frac{nT + a}{\overline{n}_c T + a} q_n \tag{5.3.25}$$

特に，多数回の横断において，近似的に $q_n$ は $n$ 個の要求をサービスする横断の割合となる．それぞれそのような横断はきっかり $t_n$ 時間単位を必要とするから，$n$ 個の要求をサービスする横断で費やされる時間の割合としては $t_n q_n / \sum_{n=0}^{\infty} t_n q_n$ であるという結論に達する．$t_n$ に代入を施せば，ただちに式(5.3.25)が得られる．

式(5.3.25)を式(5.3.18)に代入すると，次式を得る．

$$\begin{aligned}\bar{w}(x) &= \frac{a}{2} + \frac{1+\lambda T}{2(\overline{n}_c T + a)} \sum_{n=0}^{\infty} {t_n}^2 q_n \\ &= \frac{a}{2} + \frac{1+\lambda T}{2(\overline{n}_c T + a)} \sum_{n=0}^{\infty} [T^2 n^2 q_n + 2aTnq_n + a^2 q_n]\end{aligned}$$

式(5.3.23)と(5.3.24)で与えられるモーメントを用いると，代入と簡単化の結果，次式のようになる．

$$\bar{w}(x) = \frac{a + \lambda T^2/2}{1 - \lambda T} \tag{5.3.26}$$

---

↘ すなわち，式(5.3.23)が得られた．
式(A)をさらに微分して2階の導関数を求めると，

$$\begin{aligned}\frac{d^2Q(z)}{dz^2} &= \lambda a \frac{dQ(z)}{dz} + \lambda^2 T(a+T) e^{-\lambda(a+T)(1-z)} \frac{dQ(w)}{dw} \\ &\quad + (\lambda T)^2 e^{-\lambda(a+2T)(1-z)} \frac{d^2Q(w)}{dw^2} \qquad (w = e^{-\lambda T(1-z)})\end{aligned}$$

極限を取ると，

$$\begin{aligned}\lim_{z\to 1} \frac{d^2Q(z)}{dz^2} &= \lim_{z\to 1} \lambda a \frac{dQ(z)}{dz} + \lim_{z\to 1} \lambda^2 T(a+T) e^{-\lambda(a+T)(1-z)} \lim_{w\to 1} \frac{dQ(w)}{dw} \\ &\quad + \lim_{z\to} (\lambda T)^2 e^{-\lambda(a+2T)(1-z)} \lim_{w\to 1} \frac{d^2Q(w)}{dw^2}\end{aligned}$$

$\lim_{z\to 1} \frac{dQ(z)}{dz} = \overline{n_c}$, $\lim_{z\to 1} \frac{d^2Q(z)}{dz^2} = \overline{n_c^2} - \overline{n_c}$ であるから，

$$\{1 - (\lambda T)^2\}(\overline{n_c^2} - \overline{n_c}) = \{\lambda a + \lambda^2 T(a+T)\} \overline{n_c}$$

したがって，

$$\begin{aligned}\overline{n_c^2} &= \frac{1 + \lambda a(1+\lambda T)}{1 - (\lambda T)^2} \overline{n_c} \\ &= \frac{1}{1 - (\lambda T)^2} \overline{n_c} + (\overline{n_c})^2\end{aligned}$$

を得る．この式を少々変形すると，本文中の式(5.3.24)になる．

これがSCANシステムに対する式(5.3.16)と比較するべき表現である†．主要な問題点は，全ての$x(0 \leq x \leq 1)$に対して，$\bar{w}(x)$がSCANシステムにおけるよりもFSCANシステムにおいて大きいことである．

## 5.4 相互に作用し合う入出力とCPU待ち行列のモデル

オペレーティング・システムを構成するプログラムを活性化することによって発生するプロセスについての研究は，完全な一般性を具えることを意図するならば，相互に作用し合う待ち行列のネットワークに触れずには済ますことができない．諸プロセッサや入出力諸装置等の割当て，さらにはデータ・ベースへの到達等の機会を待つジョブの前にはこのような待ち行列のネットワークが存在しているからである．待ち行列から成るそのようなシステムに対して有効なモデルは，往々にして，限定的な諸仮定——その主なものに指数分布するサービス時間がある——を設けることなくしては解析を全うすることが困難である．そこで，本節では，指数サービス時間分布を仮定し，CPUと入出力の相

---

**†訳注**　式(5.3.26)と式(5.3.16)を比較するのは不当である．

式(5.3.16)の$\bar{w}(x)$は，平衡状態にあるSCANシステムをランダムに選んだ任意の時刻$t$に到着したトラック$x$への要求から観て，時刻$t$からヘッドが$x$に到る時刻$t'$までの待ち時間$t'-t$の平均ではなく，それは，任意に選んだ走査に対して取られた平均である．その際，$n$番目の走査(横断)が選ばれる確率は，全ての$n$に対して互いに等しいとされる．

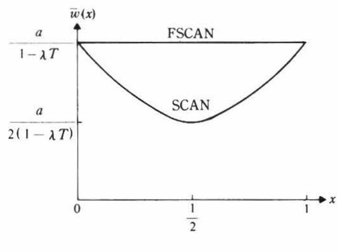
したがって，$a \to 0$を考えると，式(5.3.16)では$\bar{w}(x) \to 0$となるのに，式(5.3.26)では$\bar{w}(x) \to (\lambda T)^2/2(1-\lambda T)\ (\neq 0)$である結果になるのは当然である(FSCANに対する$\bar{w}(x)$の定義がSCANに対するそれと同じであるならば，FSCANに対する$\bar{w}(x)$も，$a \to 0$のとき，0に収束しなければならない)．

両方式を比較するためには，両方式とも同じ定義の$\bar{w}(x)$を用いなければいけない．式(5.3.16)と比較するべきFSCANに対する$\bar{w}(x)$を求めると，

$$\bar{w}(x) = \frac{a}{1-\lambda T} \qquad (5.3.26)'$$

である．

比較の結果は，下図の通り，両端($x=0, 1$)で両者は等しく，両端を除く全ての$x(0<x<1)$に対しては，FSCANに対する$\bar{w}(x)$がSCANに対するそれより大であり，しかも，$x=1/2$では，SCANに対する$\bar{w}(x)$はFSCANに対するそれの1/2であるということになり，さらに，FSCANでは差別待遇がないが，SCANでは内部のトラックに対する優遇が見られるということである．



互作用の研究にとって特に参考とするに相応しい題材として閉鎖循環型待ち行列システムを選んで，簡単にではあるが，考察することにする．

システム内に存在するジョブの総数が一定値(たとえば，正整数 $N$)に保たれるとき，システムは**閉じて**いる(または，**閉鎖型**である)という．このようなシステムの解析は有限マルコフ連鎖——状態は各待ち行列へのジョブの**配置状況**(configuration)を表現する——を用いて行われることがしばしばである．しかしながら，不幸にして，あまり大きくはない $N$ の値に対してすら状態(配置状況)の数は厖大なものとなり，その結果，解析に要する費用† は極めて大きいものになりがちである．計算機を利用した興味ある求解システムが Wallace，他 [14, 15] によって述べられている． このシステムは，有限マルコフ連鎖，特に本章と前章の諸応用において生じるような有限マルコフ連鎖の定式化と求解を容易にし，しかもある程度の自動化をも行っている．

まず，**図 5.4-1** に示される単純な**循環型待ち行列**(cyclic queue)モデル [16] を考察しよう．そこでは単一の CPU と I/O プロセッサが示されている．ジョブは CPU 待ち行列に加わることによりシステムに入るが，それが生じるのはあるジョブがシステムから退去する時点においてだけである．それゆえ，システム内のジョブ数は一定値 $N$ に保たれる． CPU でサービスを受けた後， ジョブは定常確率 $\alpha$ でシステムから離脱し(そうすると新しいジョブがただちに CPU 待ち行列に加わる)，確率 $1-\alpha$ で I/O 待ち行列に加わる．I/O 待ち行列は普通の FIFO 式待ち行列であって，そこから退去するものはただちに CPU 待ち行列に加わる． 各 CPU サービス時間の分布は， パラメータ $\mu_1$ を持つ指数分布であり，一方各 I/O サービスのそれは，パラメータ $\mu_2$ を持つ指数分布である．一つのジョブが必要とする CPU 実行総量もまた指数分布でなければならないことに注意しよう．なぜなら，それは指数サービス時間の幾何分布に

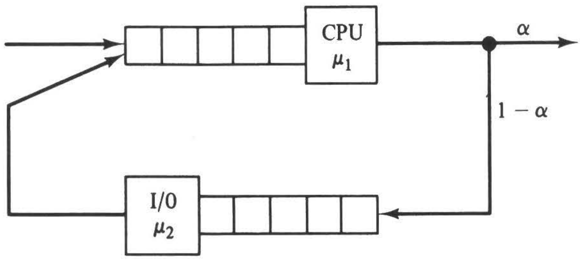



**図 5.4-1** 循環型待ち行列モデル

---

†**訳注** 費用，すなわち，計算量である．

従う和となるというモデルの仮定があるからである．ジョブがCPU待ち行列で待つ平均の回数は $1/\alpha = \sum_{i=1}^{\infty} i(1-\alpha)^{i-1}\alpha$ であるから，総CPU実行時間の分布のパラメータは $\alpha\mu_1$ である．同様の論法で必要な総I/Oサービスはパラメータ $\alpha\mu_2/(1-\alpha)$ を持つ指数分布に従うことが示される．

$X(t)$ は，時刻 $t$ においてCPU待ち行列で待機中のジョブとサービス中のものの数を表すものとする．$X(t)$ が一様なマルコフ過程であることは容易に分る．さらに，$0<\mu_1<\infty$，$0<\mu_2<\infty$，および $0\leq\alpha<1$ に対して定常分布 $\{p_i\}_{i=0}^{N}$ が存在する†．4.2節の方法を用いると，次の平衡方程式が容易に立てられる．

(5.4.1)　$$(1-\alpha)\mu_1 p_i = \mu_2 p_{i-1} \quad (i=1, 2, \ldots, N)$$

$\sum_{i=0}^{N} p_i = 1$ を満足する式(5.4.1)の解は次のように与えられる．

(5.4.2)　$$p_i = \frac{1-\rho}{1-\rho^{N+1}}\rho^i$$

ただし，

(5.4.3)　$$\rho = \frac{\mu_2}{(1-\alpha)\mu_1}$$

システムにおける平均滞在時間の尺度を得るために，まず，統計的平衡状態における引き続く退去の間の時間の期待値 $\bar{d}$ を考えよう．空でないCPU待ち行列からの退去率は明らかに $\alpha\mu_1$ である．というのは，$1/(\alpha\mu_1)$ が総サービス要求量の期待値であるからである．$1-p_0$ はCPU待ち行列が空でない定常確率であるから，平均退去率として次の式が成り立つ．

$$\frac{1}{\bar{d}} = \alpha\mu_1(1-p_0)$$

式(5.4.2)を用い，$\bar{d}$ について解くと，次式を得る．

(5.4.4)　$$\bar{d} = \frac{1}{\alpha\mu_1}\frac{1-\rho^{N+1}}{\rho-\rho^{N+1}}$$

ところで，$1/\bar{d}$ は平均到着率でもある．したがって，リトルの公式を適用すると，システムにおける平均滞在時間として次の結果を得る．

---

†訳注　$p_i = \lim_{t\to\infty} P_r[X(t)=i] \quad (i=0, 1, 2, \ldots)$

(5.4.5) $$W = \bar{d}N$$

上記の結果は，指数サービス時間を持つ待ち行列の閉じたどんなネットワークをも包含するように一般化することができる [17]．図 5.4–1 のシステムは，単に，直前に“退去”したばかりのジョブが実際には(退去せずに)CPU 待ち行列の後尾にフィードバックされるのであると想定することにより‘閉じる’ことができるということに注意しよう．この新しい閉じたシステムは，分布 $\{p_i\}_{i=0}^{N}$ が同じであるという意味で元のシステムに等価である．

一般のシステムは 1, 2, ..., $M$ と番号付けられる $M$ 個の**段**(stage)の集合で構成されている．第 $i$ 段はサービス率 $\mu_i$ を持つ $M/M/r_i$ 型待ち行列であると仮定する(4.3 節を見よ)．第 $i$ 段でサービスを完了したジョブは，システムの状態とサービスを完了したばかりのこのジョブには独立に固定されている確率 $q_{ij}$ で第 $j$ 段の待ち行列に直ちに加わる．段間遷移行列を $\mathrm{Q} = ((q_{ij}))$ で表し，$N$ はシステム内のジョブの固定された個数を表すものとする．システムの状態は $M$ 組$(n_1, n_2, \ldots, n_M)$で表される．ただし，$n_i$ は第 $i$ 段で待機中とサービス中の個数であって，次式を満足する．

(5.4.6) $$\sum_{i=1}^{M} n_i = N$$

各々が非零の遷移確率を持つような遷移の有限系列によって，システムは任意に与えられた状態から他の任意の状態へ推移することができるという通常設けられる仮定を置くことにする．したがって，対応するマルコフ過程 $X(t)$ は**既約**(irreducible)であり，唯一の定常確率分布を持つことになる．この分布を計算するための形式的方法を次に述べる議論において考察することにしよう．そこでは，解析の主要なステップのみを提供し，決まりきった(しかし，時には手の込んだ)代数演算は興味を持つ読者に任せることにする．

$P(n_1, n_2, \ldots, n_M)$ は，状態$(n_1, n_2, \ldots, n_M)$の平衡状態確率を表すものとする．4.2 節の方法を用いると，次の平衡方程式が得られる(図 5.4–2 参照)．

(5.4.7)
$$\Big[\sum_{k=1}^{M} \delta(n_k)\alpha_k(n_k)\mu_k\Big] p(n_1, \ldots, n_M) = \sum_{j=1}^{M}\sum_{k=1}^{M} \delta(n_k)\alpha_j(n_j+1)\mu_j q_{jk} p(n_1, \ldots, n_j+1, \ldots, n_k-1, \ldots, n_M)$$

ただし，

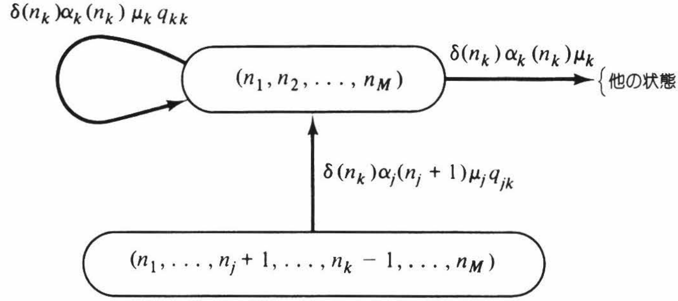



**図 5.4-2**　一般待ち行列ネットワークにおける隣接する状態

$$\delta(n_j) = \begin{cases} 0 & (n_j = 0) \\ 1 & (n_j > 0) \end{cases}$$

また，

(5.4.8)
$$\alpha_k(n_k) = \begin{cases} n_k & (n_k \leq r_k) \\ r_k & (n_k \geq r_k) \end{cases}$$

‘フィルタ’ $\delta(n_i)$ の目的は，空な待ち行列($n_i = 0$)からの退去に対応する項を排除することにある．関数 $\alpha_k(n_k)$ は，単に，第 $k$ 段にあるジョブの総数が $n_k$ であるときその段でサービス中であるジョブの個数を与える．いつものように，式(**5.4.7**)は，一つの状態から出る流れとその状態へ入る流れの平衡を取っているものと解釈できよう．はっきりと述べると，式(**5.4.7**)の左辺は状態($n_1$, $n$, ..., $n_M$)からの遷移率を与え，右辺は状態($n_1$, $n_2$, ..., $n_M$)への遷移率を与えている．

式(**5.4.7**)に対して変数分離形(または，積形式)を取る解を見出す試みは下記の方法でうまく行く．まず，数

(5.4.9)
$$\beta_i(n) = \begin{cases} n! & (n \leq r_i) \\ r_i^{n-r_i} \cdot r_i! & (n \geq r_i) \end{cases}$$

を定義する．次に，下記の仮定解を考える．

(5.4.10)
$$p(n_1, \ldots, n_M) = \frac{1}{G(N)} \prod_{i=1}^{M} \frac{x_i^{n_i}}{\beta_i(n_i)}$$

ただし，$G(N)$ は正規化パラメータであり，パラメータ $x_i$ は後で決定するべきものである．式(**5.4.10**)を式(**5.4.7**)に代入し，$\beta_i(n+1) = \alpha_i(n+1)\cdot$

$\beta_i(n)$ であることに注意して，ある程度の代数演算を施すと，次のことが明らかになる．すなわち，式(5.4.10)が式(5.4.7)を満足する確率分布であるのは，$\mathbf{x}' = (\mu_1 x_1, \mu_2 x_2, \dots, \mu_M x_M)$ が次の連立 1 次方程式

$$\mathbf{x}' = \mathbf{x}'\mathrm{Q} \tag{5.4.11}$$

を正規化条件

$$G(N) = \sum_N \left\{ \prod_{i=1}^{M} \frac{x_i^{n_i}}{\beta_i(n_i)} \right\} \tag{5.4.12}$$

の下で満足するときであり，かつそのときに限る．ここで $\sum_N$ は $n_1 + \cdots + n_M = N$ であるような全ての状態$(n_1, n_2, \dots, n_M)$についての総和を表す．解の存在と一意性は行列 Q の既約性によって保証される．この既約性は，マルコフ過程 $X(t)$ が既約であるという最初の仮定に由来する [17]．要約して述べると，以上のような訳で，任意の閉じた指数型のシステムに対する確率分布は，連立方程式(5.4.11)を解くことにより，つまり，行列 Q の固有ベクトル問題を解くことにより計算することができる．

多くの興味ある構造がこれらの結果を用いて研究できるであろう．いわゆる**セントラル・サーバ型ネットワーク**(central-server network) [18] は以下に述べるようにその一例である．(他の例については演習問題 5.10 を見よ．)

セントラル・サーバ型ネットワークは図 5.4-3 に示されている．そこでは，段 1 がプロセッサであり，段 2 から $M$ までは入出力装置であると見なされる．図 5.4-3 における流れの方向に対応して行列 Q は次式で与えられる．

$$\mathrm{Q} = \begin{pmatrix} q_1 & q_2 & \cdots & q_M \\ 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ 1 & 0 & \cdots & 0 \end{pmatrix} \tag{5.4.13}$$

ただし，$q_i$ は，プロセッサから退去するジョブが次に第 $i$ 段へ進む確率である．この場合の式(5.4.11)は，式(5.4.13)で与えられる Q を持ち，容易に解が得られる連立 1 次方程式である．特に，$(z, q_2 z, \dots, q_M z)$の形の任意のベクトルは式(5.4.11)を満足する．$z$ は自由に選べるから，$z = \mu_1$ を選択するのが便利である．したがって，$x_1 = \mu_1/\mu_1 = 1$ ということを使い，$r_i = 1$ は全ての

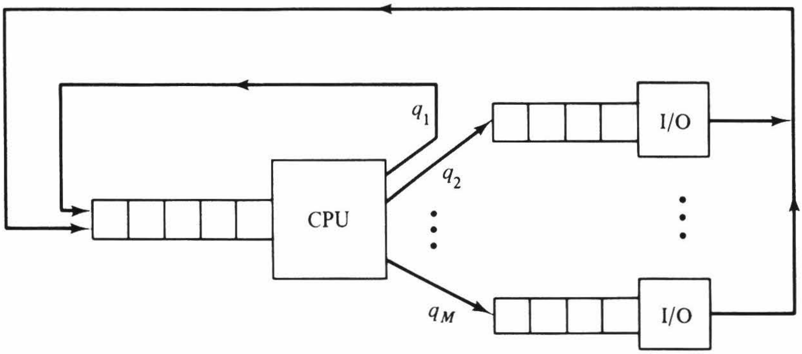



図 5.4-3　セントラル・サーバ　モデル

$n$ に対して $\beta_i(n)=1$ が成り立つことを意味するということに注意すると，式(5.4.10)から次式を得る．

(5.4.14)
$$p(n_1, \ldots, n_M)=\frac{1}{G(N)}\prod_{i=2}^{M}\left(\frac{q_i\mu_1}{\mu_i}\right)^{n_i}$$

ただし，

(5.4.15)
$$G(N)=\sum_{N}\prod_{i=2}^{M}\left(\frac{q_i\mu_1}{\mu_i}\right)^{n_i}$$

式(5.4.15)で総和を取るべき項に $n_1$ が現れないことに注意しよう．したがって，和を次のように表現することもできる．

(5.4.16)
$$G(N)=\sum_{\mathbf{n}\in\Lambda(N)}\prod_{i=2}^{M}\left(\frac{q_i\mu_1}{\mu_i}\right)^{n_i}$$

ここで，$\Lambda(N)$ は，$\sum_{i=2}^{M} n_i \leq N$ となるような状態 $\mathbf{n}=(n_1, n_2, \ldots, n_M)$ の集合である．$G(N)$ に対する後者の総和法は下記で再び用いられる．本節は，セントラル・サーバ型ネットワークにおける平均待ち行列長の表現式を求めることで締括ることにする．

$\overline{n}_j$ は第 $j$ 段で待機中とサービス中のジョブの定常状態における平均個数を表すものとすると，次のように書くことができる．

(5.4.17)
$$\overline{n}_j=\sum_{k=1}^{N}\bar{P}_j(k)$$

ただし，$\bar{P}_j(k)=\Pr[n_j \geq k]$ は，第 $j$ 待ち行列の長さが $k$ より大きいか等し

い平衡状態確率を表す．$\bar{P}_j(k)$ の計算において，状態 $\mathbf{n}=(n_1, n_2, \ldots, n_M)$ の平衡状態確率を $p(\mathbf{n})$ と書き，さらに集合

(5.4.18) $$\Lambda_j(k)=\left\{\mathbf{n}\,\middle|\,\sum_{i=2}^{M} n_i \leq N \text{ かつ } n_j \geq k\right\}$$

を定義しておくと便利である．

さて，プロセッサ段に対しては次の式が成り立つ．

$$\bar{P}_1(k)=\sum_{\mathbf{n}\in\Lambda(N-k)} p(\mathbf{n})=\sum_{\mathbf{n}\in\Lambda(N-k)} \frac{1}{G(N)} \prod_{i=2}^{M}\left(\frac{q_i\mu_1}{\mu_i}\right)^{n_i}$$

これに式(5.4.16)を適用すると次式を得る．

(5.4.19) $$\bar{P}_1(k)=\frac{G(N-k)}{G(N)}$$

$j=2, 3, \ldots, M$ に対しては次の式が成り立つ．

$$\bar{P}_j(k)=\sum_{\mathbf{n}\in\Lambda_j(k)} p(\mathbf{n})$$

これは次の形に置き直せる．

(5.4.20) $$\bar{P}_j(k)=\frac{1}{G(N)}\left[\frac{q_j\mu_1}{\mu_j}\right]^k \sum_{\mathbf{n}\in\Lambda(N-k)} \prod_{i=2}^{M}\left(\frac{q_i\mu_1}{\mu_i}\right)^{n_i}$$

それゆえ，式(5.4.16)と(5.4.19)を用いると次式を得る．

(5.4.21) $$\bar{P}_j(k)=\left[\frac{q_j\mu_1}{\mu_j}\right]^k \bar{P}_1(k)=\left[\frac{q_j\mu_1}{\mu_j}\right]^k \frac{G(N-k)}{G(N)}$$

結局，式(5.4.17)へ代入することにより次の結果が齎される．

(5.4.22) $$\bar{n}_1=\sum_{k=1}^{N}\frac{G(N-k)}{G(N)}$$

(5.4.23) $$\bar{n}_j=\sum_{k=1}^{N}\left[\frac{q_j\mu_1}{\mu_j}\right]^k \frac{G(N-k)}{G(N)} \quad (j=2, 3, \ldots, M)$$

多項式

(5.4.24) $$F(x)=\sum_{k=1}^{N} x^k \frac{G(N-k)}{G(N)}$$

を定義すれば，式(5.4.22)と(5.4.23)は次のように簡潔に書き表される．

$$\overline{n}_1 = F(1)$$

$$\overline{n}_j = F\left(\frac{q_j \mu_1}{\mu_j}\right) \quad (j>1)$$

したがって，まず数 $G(0), G(1), ..., G(N)$ を計算し，次にこれを用いて式(5.4.24)を評価するという手順を踏むことにより平均待ち行列長が算出されることになる．

Lewis と Shedler の論文[19]は本節のモデルに関連がある．その論文では，CPU 利用率とシステム経費について研究を進めることを主目的として，ページング・システム(第6章参照)の待ち行列モデルが考案されている．そこでのモデルは，本節で採用したモデル程指数分布の仮定に依存してはいない．しかし，その反面では構造が循環型待ち行列に限定されている．その解析と結果はどちらかといえば手の込んだものであるので，そうした理由から本節にそれを含めることはしなかった．

## 5.5　緩衝記憶割当て問題

システム設計において生じる非常に重要な問題のいくつかは，緩衝記憶技法の構造とパラメータの指定に関するものである．**緩衝記憶**(buffer storage)の必要性は少なくとも次の三つの局面において生じる．1)補助記憶または入出力装置に対する出力前および入力後の情報の**蓄積**(storage)，2)活性タスク間で通信されるメッセージとデータの蓄積，3)タイムシェアリング・システムにおけるように遠隔端末から到来するかあるいはそこへ送出されるメッセージの蓄積である．

**緩衝領域**(buffer)設計問題に潜む問題点は，同期について2.6節で論じた緩衝領域問題によって説明することができる．その最も単純な形態は，1回に1個の出力メッセージを生成する1個の循環タスク，これらのメッセージを置く $N$ 個のセルから成る緩衝領域，およびメッセージの生成に対して非同期的に実行しつつ，緩衝領域からある出力装置へ送出する1個の循環出力タスクから成る．このシステムの設計において最初に生じる問題の一つは，固定ブロックの記憶領域を緩衝領域として割当てると仮定して，N の値をどう定めるかである．記憶領域利用率と**処理率**(throughput rate)の間には明らかに釣合い関係が存在していて，一方の改善は他方の低落を招く．

もっと明確にするため，続いてメッセージを生成するのに必要な時間は互いに独立であり，かつパラメータ $\lambda$ を持つ指数分布に従うものと仮定しよう．到

着するメッセージによって緩衝領域が一杯になったときは常に，メッセージ入力源は緩衝領域のセルが使用可能になるまで機能を停止すると仮定する．メッセージを緩衝領域から取除いた後それを出力するのに要する時間は，平均が $1/\mu$ の指数分布に従う確率変数であると仮定しよう．緩衝領域内で待っているメッセージと出力中のメッセージの総数でシステムの状態を定義すると，4.3 節におけるモデルを使うことができる．特に，式(4.3.8)からメッセージの平均個数が得られる．すなわち，

$$\bar{B}_N=\sum_{n=0}^{N}\frac{n\rho^n(1-\rho)}{1-\rho^{N+1}}=\frac{\rho}{1-\rho}-\frac{(N+1)\rho^{N+1}}{1-\rho^{N+1}} \tag{5.5.1}$$

ただし，

$$\rho=\frac{\lambda}{\mu}$$

メッセージの処理率は，$\lambda$ と入力源が機能している確率との積となる．すなわち，

$$H_N=\lambda(1-p_N) \tag{5.5.2}$$

ただし，式(4.3.8)から

$$p_N=\frac{(1-\rho)\rho^N}{1-\rho^{N+1}}$$

これらの結果から，処理率 $H_N$ と記憶領域利用率 $\bar{B}_N/N$ の間の釣合い関係を学び取ることができる．

上記の例が示すように，待ち行列理論は，本来，多くの緩衝記憶問題の解析に役立つものである．しかしながら，現実主義からの関心と数学的な扱い易さはしばしば相反するものである．たとえば，難しいモデルへ導く仮定に**一括**(bulk)到着(演習問題 4.2 と 5.5 を見よ)と一括サービスがある．他方，緩衝領域の大きさが有限であるという通常の仮定を伴えば，有限状態システムが得られる．こうして，マルコフ連鎖解析で生じる有限連立一次(平衡)方程式を解くことによって，望みの工学的結果をしばしば得ることができる．

多くの可変長メッセージが同一の記憶領域を共用しなければならないときに生じるもう一つの重要な問題は，**動的記憶割当て法**(dynamic storage allocation)と**固定分割法**(fixed partitioning method)のどちらを採用するかを決定することである．特に，与えられた緩衝記憶領域へメッセージを給送する入

力線はM本存在すると仮定しよう．固定分割方式においては，各入力に対してそれぞれ別々の固定長ブロックの記憶領域が予め用意される．ブロックの大きさはメッセージ送出源からの**通信量**(traffic)に依存して決められよう．**溢れ**(overflow)が生じる場合には，追加の記憶領域を用意するか，それとも緩衝領域内に十分な大きさの空間が得られるようになるまでそれ以上の通信を行わないように送出源を一時的に停止させるかする機構の存在を仮定しなければならない．この記憶割当て方法の不利な点は，主として飽和状態にならないブロック内の使える空間が溢れを生じているブロックのために利用可能になっていないときに現れる記憶領域利用率の潜在的な低さにある．しかし，他の割当て手続きと比較するならば，固定分割法は簡素であるという利点がある．

動的方法または**連結リスト法**(linked-list method)による記憶領域割当てでは，各メッセージは緩衝領域内の鎖状に連結したセルに記憶されるのが普通である[7]．あるメッセージが緩衝領域から取除かれると，それが占めていた空間は常に**使用可能空間リスト**(available space list)に戻される．新しいメッセージが到着すると，それは使用可能空間リストの先頭から取られる適当な数のセルに記憶される．一般に，ここでも対溢れ機構の存在を仮定しなければならない．また，各入力線に対する同一送出源からのメッセージの待ち行列の形成と**行列頭ポインタ**(head-of-queue pointer)の作成を行う定形的な技法が存在することも仮定されなければならない．

連結割当ての明白な不利益は連結のために取っておかなければならない空間にある．これに関連して，各セルまたはブロック内の文字数 $b$ を変化させることによって釣合いを取るようにすることができる．各セルにおいては，連結用に一定の $c$ 文字分を当てるものとすると，$b-c$ 文字がメッセージの記憶用に使用可能である．今，長さ $m$ がセルの大きさ $b$ に比べて適度に大きいメッセージの記憶を考えるものとする．1次近似としては，メッセージによって占められるセルの個数は $m/(b-c)$ であり，最後のセル内で未使用の空白部分の大きさは $(b-c)/2$ である．したがって，所与のメッセージの記憶の結果，他のメッセージの記憶のために($m$ 以外に)使用不能になる空間の全体は，近似的に次のようになる．

$$w_m = \frac{b-c}{2} + \frac{mc}{b-c} \tag{5.5.3}$$

したがって，$w_m$ を最小にする $b$ の値は，微分法によって次のように求めることができる．

$$b_{\text{opt}} = c + \sqrt{2\,mc} \tag{5.5.4}$$

もしメッセージの長さが確率変数であるならば，上式中の $m$ の値としては期待値を使うことができる．この問題に対し，さらに念入りなもっとよい近似解析を行うこともできる[20]が，$b_{\text{opt}}$ の関数としての依存関係は本質的に(式(5.5.4)と)変らない．

割当て方策自体は一先ず考慮外に置き，今，上記のメッセージ緩衝記憶モデルを，テレタイプ，タイプライター等が直接に中央計算機システムと通信する**低速データ伝送**(low-data-rate)システムの大きなクラスに対応するように特殊化しよう[20]．目標は，ここでも，緩衝領域の大きさの関数として緩衝記憶の性能を解析することである．**図 5.5-1** に示されるように，$M$ 本の入力線があって，それぞれ**伝送速度**(transmission rate)は $1/\alpha$ であると仮定する．便宜上，$\alpha$ は1文字を緩衝記憶に搭載するのに必要な時間(秒を単位とする)を与えるものと仮定する．1本の入力線上の活動の統計的な特徴付けを考えることから始める．

低速データ伝送の仮定は，緩衝記憶内で1文字ずつメッセージを組立てて行くための時間は，緩衝領域からメッセージをプログラムが読取るのに要する時間よりも極めて長いということを意味している．実際，ここでは，入力速度の時間尺度で見ると，緩衝領域からメッセージを読取る過程は実効的に一瞬間に過ぎないと仮定する．したがって，もし一つの送出源からのメッセージによって占められる空間に注目するとするならば，**図 5.5-2** に示す実現例のような関数を持つ過程 $Z(t)$ を得ることになる．$Z(t)$ は実際には区間 $X_n$ 内で(1度に1文字づつ)階段的に増大して行くのであるが，この増大は直線で近似されて

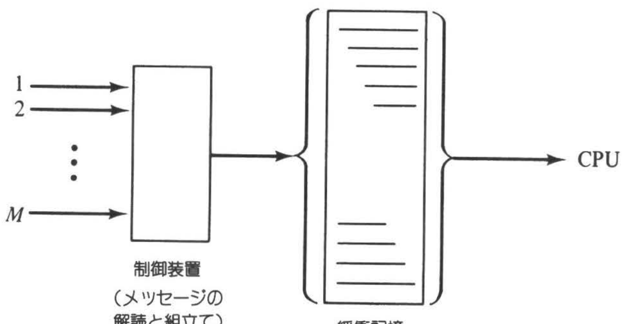



**図 5.5-1** 入力緩衝記憶のモデル

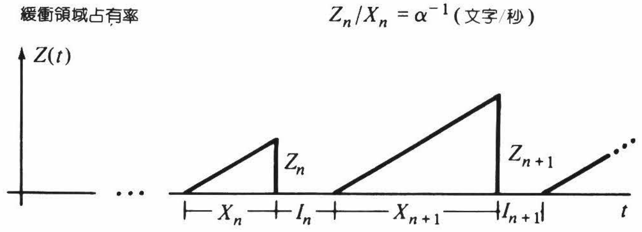



**図 5.5-2**　緩衝領域占有過程

いる．解析を容易にするため，メッセージの大きさ $Z_n$(したがってまた，伝送区間 $X_n$ もそうであることになるが) $(n=1, 2, \ldots)$ は統計的に互いに独立であると仮定する．$X_n$ の定常分布は

$$F(x) = \Pr[X_n \leq x] \quad (n=1, 2, \ldots)$$

であるとし，その $i$ 次のモーメントを $E(X^i)$ と書く．簡単のため，$F(x)$ は連続関数であると仮定し，この仮定は妥当な1次近似であると認めることにする．得られる結果を任意の分布に適用できるように改訂することは容易である．

遊休区間 $I_1, I_2, \ldots$ は，互いに独立な等しい分布に従う確率変数の系列を作る．その各々は $X_n (n=1, 2, \ldots)$ には統計的に独立である．遊休期間分布の $i(i \geq 1)$ 次のモーメントを $E(I^i)$ と書く．いわゆる**交代更新過程**(alternating renewal process) $Z(t)$ は，長さがある適当な確率分布に従う遊休期間 $I_0$ から始まると仮定する．ここでは平衡状態における振舞に関心があるので，初期条件に重要性は認められない．

さて，長時間走行における平均緩衝領域占有率

$$\bar{B} = \lim_{t \to \infty} \mathrm{E}(Z(t)) \tag{5.5.5}$$

の計算を進めることにする．まず，$Z(t)$ の分布は混合型でなければならないことに着目する．遊休期間が存在するから，$Z(t)=0$ に集中する離散的な確率塊がある．ランダムな時点 $t$ での観察を考えよう．緩衝領域が空である状態に対応する離散的成分を分離すると，次のようになる．

$$\Pr[Z(t)=0] = \Pr[t \text{ は遊休期間に含まれる}] \tag{5.5.6}$$

また $x>0$ に対しては次の通りである．

(5.5.7) $\Pr[Z(t) \leq x] = \Pr[Z(t) = 0] + \Pr[t$ は伝送区間内にあり，区間の経過時間は $\alpha x$ を越えていない$]$

式(5.5.6)の極限値と統計的平衡状態において遊休期間に費やされる時間の割合とを同一視すると，次式が成り立つ(p. 265 の脚注を参照せよ).

$$\lim_{t\to\infty} \Pr[Z(t) = 0] = \frac{\mathrm{E}(I)}{\mathrm{E}(I) + \mathrm{E}(X)} \tag{5.5.8}$$

$t$ が伝送区間内に落ちるという条件を付けると，$t$ までのこの区間の経過時間の分布は，系列 $X_1, X_2, \ldots$ から成る更新過程における**後退再帰時間**(backward recurrence time)の分布によって与えられる(補遺 B を参照せよ). したがって，経過時間が $\alpha x$ より小さいか等しい確率 $P_e(\alpha x)$ は次式で与えられる.

$$P_e(\alpha x) = \frac{1}{\mathrm{E}(X)} \int_0^{\alpha x} [1 - F(u)]\,du \tag{5.5.9}$$

$t$ が伝送区間内に落ちる極限確率を乗じると，次のようになる.

$$\lim_{t\to\infty} \Pr[Z(t) \leq x] = \frac{\mathrm{E}(I)}{\mathrm{E}(I) + \mathrm{E}(X)} + \frac{\mathrm{E}(X)}{\mathrm{E}(I) + \mathrm{E}(X)} P_e(\alpha x)$$

すなわち，

$$\lim_{t\to\infty} \Pr[Z(t) \leq x] = \frac{1}{\mathrm{E}(I) + \mathrm{E}(X)} \left[ \mathrm{E}(I) + \int_0^{\alpha x} [1 - F(u)]\,du \right] \quad (x>0) \tag{5.5.10}$$

式(5.5.8)と(5.5.10)で与えられる分布の平均が式(5.5.5)の平均緩衝領域占有率を与える．すなわち，次式が得られる.

$$\bar{B} = \frac{\mathrm{E}(X^2)}{2\,\alpha[\mathrm{E}(I) + \mathrm{E}(X)]} \tag{5.5.11}$$

また，式(5.5.8)と(5.5.10)から，大きさ $N$ の緩衝領域における溢れの長時間走行確率として次式を得る.

$$R(N) = \frac{1}{\mathrm{E}(I) + \mathrm{E}(X)} \int_{\alpha N}^{\infty} [1 - F(u)]\,du \tag{5.5.12}$$

これは，統計的平衡状態において過程 $Z(t)$ が水準 $N$ を越えている時間の割合の期待値を表してもいる．所与の水準より上にある時間の期待値に加えて，

溢れ(上限突破)事象の頻度もまた重要であり，余分の記憶領域を緩衝のために使用可能にしなければならなくなる度毎に並々ならない経費が掛かるときには(通常その通りであるが)特にそうである．この性能の尺度について値を計算するため，遊休期間—伝送の各循環に高々1度の上限突破しか起こらないこと，またこの事象は各循環内において確率 $1-F(\alpha N)$ で発生することに注意すると，**平均上限突破率**(average upcrossing rate)として次式が得られる．

(5.5.13)
$$\bar{U} = \frac{1 - F(\alpha N)}{\mathrm{E}(I) + \mathrm{E}(X)}$$

さて，それぞれが上記で仮定として与えられたかまたは導出されたような特性を持つ $M$ 個の統計的に独立な入力源によって共用される $N$ セルの緩衝領域を考えよう．動的記憶割当て方式を用いるものとする．$Z_j(t)$ は，$j$ 番目の入力源に対する緩衝領域占有過程を表す．$M$ は適度に大きいものと仮定すると，和の過程の分布

(5.5.14)
$$Z_S(t) = \sum_{j=1}^{M} Z_j(t)$$

は，中心極限定理の適用を受けて，下記の密度関数を持つ正規分布により近似され得る．

(5.5.15)
$$g(x) = \frac{1}{\sqrt{2\pi\sigma_S^2}} e^{-(x-m_S)^2/2\sigma_S^2}$$

モーメントは，$M$ 個の入力源について和を取れば式(5.5.8)と(5.5.10)の分布から得られる．$j$ 番目の入力源に対して，$\mathrm{E}(X_j^i)$ が伝送時間の $i$ 次モーメントを表し，$\mathrm{E}(I_j)$ が平均遊休時間を表すとすると，次の二つの式が成り立つ．

(5.5.16)
$$m_S = \frac{1}{2\alpha} \sum_{j=1}^{M} \frac{\mathrm{E}(X_j^2)}{\mathrm{E}(I_j) + \mathrm{E}(X_j)}$$

および

(5.5.17)
$$\sigma_S^2 = \sum_{j=1}^{M} \left[ \frac{\mathrm{E}(X_j^3)}{3\alpha^2[\mathrm{E}(I_j) + \mathrm{E}(X_j)]} \right] - m_S^2$$

したがって，共用緩衝領域における溢れの確率は次のようになる．

(5.5.18)
$$R_S(N) = \frac{1}{\sqrt{2\pi}} \int_{(N-m_S)/\sigma_S}^{\infty} e^{-u^2/2} du$$

動的な場合の結果は，特定の実現，さらにはその最も関心の対象となる運用

上の諸条件に応じて，種々な方法で固定分割システムのそれと比較することができる．たとえば，それぞれの方式において'溢れ'た文字の平衡状態における平均個数を問うこともできよう．固定分割の場合では，これは各入力源に対するブロックの大きさを超過する平均量の $M$ 倍であろうから，式(5.5.8)と(5.5.10)で与えられる分布から求めることができる．動的記憶割当てでは，それは緩衝領域全体を $Z(t)$ が超過する平均量であり，式(5.5.15)から計算できる．

公平な比較を得るためには，特殊文字(たとえば，連結のために取った空間)とメッセージにより占められる最後のブロック内で無駄になる余白によって費やされる少なからぬ量の記憶領域を考慮に入れる必要があろう．このための一つの簡単な近似的接近法は，既知のメッセージ長分布から式(5.5.3)の確率変数の分布を計算し，上記の解析にこの分布を用いることである．

前にも注意したように，もう一つの重要な性能尺度は，公称の緩衝領域の大きさを超過する頻度である．$Z(t)$ を正規分布と仮定すると，この形の結果はガウス過程の古典理論から求めることができる．一つの有益な取扱いの例が演習問題(問題5.9)で与えられる．そこでの取扱いと共に本節の解析は Gaver と Lewis [20] の結果に依っている．なお，この論文は Delgavis と Davison [21] の結果を拡張したものである．

## 演 習 問 題

5.1 任意に与えられた時刻に，セクタ当り高々1個の要求がサービスを待ちつつある(かまたはそのときサービス中である)と仮定して，ページングドラム・システムの解析を行え．ある与えられた要求が退去した後，同じセクタに次の要求が到着するまでの時間はパラメータ $\lambda$ を持つ指数分布に従うものと仮定せよ．ドラム1回転当りサービスされる要求の平均数と平均システム内滞在時間を求めよ．

5.2 5.3節で論じた FSCAN システムを，各走査の開始時に待ち行列から取られる要求の個数が最大 $N$ であるように制限にする，すなわち，各走査においてサービスされる数は，走査開始時に待っている個数を $i$ とするとき，$\min\{i, N\}$ であると定義する．この新しいシステムを NSCAN と呼ぶ．待ち行列には常に少なくとも $N$ 個の要求があり，各走査毎に丁度 $N$ 個ずつ要求がサービスされるという仮定の下で，つまり飽和状態で動作している NSCAN システムの振舞を調べよ．各走査においてヘッドは全ト

ラックを横切るという簡単化仮定を設けないことを除いて**5.3**節の諸仮定と記号を用いよ．したがって，到達する最初と最後のトラックはFSCANについて最初に記述した通りに定められる．走査時間の期待値は，$\mathrm{E}(S)=NT+(\bar{u}+\bar{v})a$ で与えられる．ただし，$\bar{u}$ は最初の要求に到るまでにヘッドが動かなければならない距離の期待値であり，$\bar{v}$ はサービスされる最初の要求から最後の要求までヘッドが動かなければならない距離の期待値である．トラック番地を単位区間からランダムに取ると仮定して，$N$, $T$, およびヘッドがディスク全面(長さ**1**の区間)を横断するのに要する時間 $a$ を用いて $\mathrm{E}(S)$ の詳細な表現式を求めよ．

**5.3**　**5.3**節におけるSCAN手続きに対して，FSCAN式順序付けをSCAN式順序付けに併合するような一般化を定義することができる．**5.3**節の数学的モデルにおいて，単位区間を $N$ 個の等しい小区間に分割するものとしよう．与えられた任意の走査において $i$ 番目の小区間内でサービスされる要求は，丁度ディスク・ヘッドが(左からまたは右から)この区間に進入し始めようとするときに既にシステム内に到着していて，しかも，$i$ 番目の小区間内の番地を持つ要求であるという併合規律を考える．言い換えると，$i$ 番目の小区間内のトラック番地に対する要求は，ディスク・ヘッドの現在位置よりも(これから通過することになる)前方に到着するものであるとしても，ディスク・ヘッドが現在既に $i$ 番目の小区間内に入っているならば，次の走査でサービスされなければならないということである．

$y_i$ は，統計的平衡状態において最初の $i$ 個の小区間をディスク・ヘッドが通過するのに要する時間の平均を表すものとする．**5.3**節の数学的モデルによって，次式が成り立つことを示せ．

$$y_i=\frac{ia}{N}\,\frac{[1-(N-i-1)\lambda T/N]}{(1-\lambda T)(1+\lambda T/N)}\quad(1\leq i\leq N)$$

極限 $N\to\infty$ においては，上記の結果は式(**5.3.7**)となり，また $N=1$ に対しては式(**5.3.8**)が成り立つことを確かめよ．

**5.4**　ここでは**5.3**節の結果を，ディスク上で定義される要求番地の任意の分布を包含するように拡張するのが目的である[22]．所与の到着が区間[$x$, $x+\Delta x$]内の番地を持つ確率は $f(x)\Delta x$[と位数 $(\Delta x)^2$ の項の和]で与えられるような連続的密度関数 $f(x)$ が与えられているものと仮定する．**5.3**節のモデルと方法を用いて，$\lambda T<1$ であるとき，

$$\bar{y}_r(x+\Delta x) = \bar{y}_r(x) + a\Delta x + \lambda T f(x)\Delta x \bar{z}_r(x) + [\text{位数 } (\Delta x)^2 \text{ の項}]$$

となることを示せ．ここで，$\bar{y}_r(x)$ はヘッドが位置0から位置 $x$ へ移動するのに要する時間の平衡状態における平均であり，$\bar{z}_r(x)$ は $x$ から0へ移動しかつ $x$ へ戻るのに要する時間の平均である．**リトルの公式**を用いて，次式が成り立つことを示せ．

$$\bar{z}_r(x) = \frac{2a\lambda T}{1-\lambda T}F(x) + 2ax$$

ただし，$F(x) = \int_0^x f(t)dt$ である．さらに進んで，次式が成り立つことを示せ．

$$\bar{y}_r(x) = ax + \frac{a\lambda^2 T^2}{1-\lambda T}F^2(x) + 2a\lambda T\int_0^x tf(t)dt$$

$\bar{y}_l(x)$ は平衡状態において位置1から位置 $x$ へ移動するのに要する平均時間であると定義し，同様に，$\lambda T<1$ であるとき，

$$\bar{y}_l(x) = ax + \frac{a\lambda^2 T^2}{1-\lambda T}[1-F(1-x)]^2 + 2a\lambda T\int_0^x tf(1-t)dt$$

となることを示せ．本章における接近法を用いて，次式

$$\bar{w}(x) = \frac{2a}{1-\lambda T}\left\{\left[\left(x-\frac{1}{2}\right)(1-\lambda T) + \lambda T\left(F(x)-\frac{1}{2}\right)\right]^2 + \frac{1}{4}\right\}$$

が成り立つこと，また，$f(x)$ が一様分布の密度であるとき，これは式(5.3.16)に縮退することを示せ．

**5.5** メッセージを文字の一括到着——1メッセージの全文字が実効的に到着と同時に緩衝領域内に記憶される(緩衝領域には十分な空間があるものと仮定すれば)——であると見なす緩衝記憶モデルを考察せよ．与えられたメッセージ長(一括めの大きさ)の定常分布，$N$ 文字の大きさの有限緩衝領域，および指数分布または幾何分布に従うメッセージ到着時間間隔を仮定せよ．閉じた形の結果† を得るのは一般に困難ではあるが，メッセージ長が有界であると仮定すれば，平衡状態における緩衝領域占有率の有限マルコフ連鎖解析が可能となる．

そのようなモデルを完全に定義し切るには，緩衝領域内に空のセルが $j$

†**訳注** 閉じた形の結果とは陽関数の形で表示された結果をいう．

個しかないときに $k(>j)$ 文字から成るメッセージが到着するとどうなるかについて定めておかなければならないことに特に注意せよ．たとえば，一つの可能性は，大きさが到着時に使用可能な空間の大きさを超過しているものは，(同じ意味で過大な全ての後続のメッセージ共々)メッセージ全体を切り捨ててしまうことである．すなわち，到着メッセージは，到着時に十分な空間があるとき，かつそのときに限り記憶されると定めることである．

5.6　リストのセル当りの最良の文字数について式(5.5.4)を導くような連結リスト式緩衝記憶モデルを考えよ．最後のセル内の平均文字数を左右している近似を取り除くことによってモデルを精密にしよう．メッセージ長 $m$ は，平均値が $\bar{m}$ である分布関数 $F(x)$ を持つものと仮定する．$b$ と $c$ は 5.5 節におけるように定義し，$a=b-c$ と置く．最後のセルへの溢れの期待値は次式のようになることを示せ．

$$\mathrm{E}(V)=\bar{m}-a\sum_{k=1}^{\infty}[1-F(ka)]$$

$F(x)$ が平均 $\bar{m}$ を持つ指数分布であると仮定するときは，次のようになることを確認せよ．

$$\mathrm{E}(V)=\frac{\bar{m}(e^{a/\bar{m}}-1)-a}{e^{a/\bar{m}}-1}$$

これは極限 $\bar{m}\to\infty$ においては $a/2$ となる．

5.7　**図 P 5.7** に示されるようにメッセージが順に記憶されなければならないという制約下にある緩衝記憶におけるポアソン型 FIFO 式待ち行列[23]の動作を解析したい．新しいメッセージは常に待ち行列の最後尾に置かれる．待ち行列からメッセージが取去られると，明らかに待ち行列の先頭は右へ1位置だけ移動する．しかし，待ち行列が空になったとき，次に到着

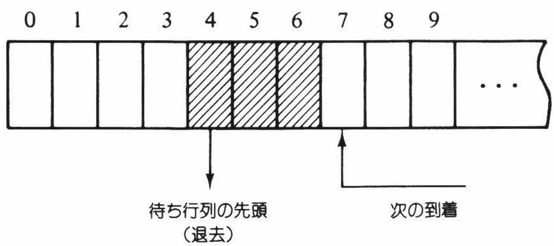



図 P 5.7

するメッセージは最初のセル位置へ置かれるものとする．こうして，待ち行列の先頭は元の場所に復帰する．簡単のため，緩衝領域の大きさは有界でないと仮定しよう．

$\pi_{ij}$ は，待ち行列の先頭の番地(相対位置)が $i$ であって，かつ待ち行列の長さが $j$ となる——たとえば，図 P 5.7 では $i=4$ で $j=3$ である——定常(結合)確率を表すものとする．$\lambda$, $\mu$, および $\rho=\lambda/\mu$ は，到着率，退去率，および利用率をそれぞれ表すものとする．$\pi_{ij}$ に関する次の平衡方程式が成り立つことを確認せよ．

$$\pi_{0j}=\frac{\rho}{1+\rho}\pi_{0(j-1)} \quad (j\geq 1), \qquad \pi_{00}=\frac{1}{\rho}\sum_{m=0}^{\infty}\pi_{m1}$$

$$\pi_{i1}=\frac{1}{1+\rho}\pi_{(i-1)2} \quad (i\geq 1)$$

$$\pi_{ij}=\frac{1}{1+\rho}\pi_{(i-1)(j+1)}+\frac{\rho}{1+\rho}\pi_{i(j-1)} \quad (i>0,\ j>1)$$

これらの方程式は次式によって満たされることを示せ．

$$\pi_{ij}=\begin{cases}\pi_{00}\left(\dfrac{\rho}{1+\rho}\right)^{j} & (i=0)\\[2ex] \pi_{00}\left[\dfrac{\rho}{(1+\rho)^2}\right]^{i}\left(\dfrac{\rho}{1+\rho}\right)^{j}\dfrac{j}{2i+j}\dbinom{2i+j}{i} & (i>0)\end{cases}$$

2変数(bivariate)母関数

$$M(z_1,\ z_2)=\sum_{i=0}^{\infty}\sum_{j=0}^{\infty}\pi_{ij}z_1{}^{i}z_2{}^{j}$$

を定義し，次式が成り立つことを示せ．

$$M(z_1,\ z_2)=\frac{\rho(1-\rho)z_2+(1-\rho)(z_2-z_1)-z_1z_2M_1(z_1)}{(z_2-z_1)+\rho(1-z_2)z_2}$$

ただし，

$$M_1(z_1)=\sum_{i=0}^{\infty}\pi_{i1}z_1{}^{i}$$

待ち行列の先頭の平均位置は

$$\lim_{z_1\to 1}\frac{dM(z_1,\ 1)}{dz_1}=\left(\frac{\rho}{1-\rho}\right)^2$$

となること，また平均待ち行列長は

$$\lim_{z_2 \to 1} \frac{dM(1,\, z_2)}{dz_2} = \frac{\rho}{1-\rho}$$

となることを導出せよ．

**5.8**　緩衝記憶読出し時間として零でない $\beta$ を許すことにより，したがって，**見本**(sample)関数 $Z(t)$ の不連続性を取り除くことによって**5.5**節のモデルを一般化することは可能である．見本関数の一例が**図 P 5.8** に示されている．そこでは，$Z'(t)$ が新しい緩衝領域占有過程を表す．次式が成立することを示せ．

$$m = \lim_{t\to\infty} \mathrm{E}(Z'(t)) = \frac{1}{2\,\alpha} \frac{\mathrm{E}(X^2)\,(1+\beta/\alpha)}{\mathrm{E}(X)\,(1+\beta/\alpha) + \mathrm{E}(I)}$$

$$\lim_{t\to\infty} \mathrm{Var}(Z'(t)) = \frac{1}{3\,\alpha^2} \frac{\mathrm{E}(X^3)\,(1+\beta/\alpha)}{\mathrm{E}(X)\,(1+\beta/\alpha) + \mathrm{E}(I)} - m^2$$

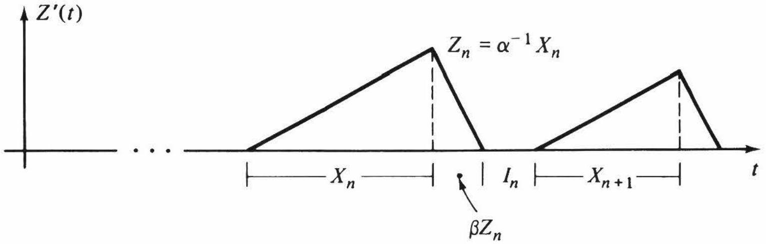



**図 P 5.8**　非零読出し時間

**5.9**　緩衝領域設計問題において興味ある性能尺度の一つは，公称の緩衝領域の大きさを超過する平均回数である．$Z'_S(t)$ は，$M$ 個の単位過程 $Z'_i(t)$ $(1 \leq i \leq M)$ の和を近似するガウス過程を表すものとし，各単位過程は演習問題5.8で定義されるものとする．一般に，水準突破の統計は，過程 $Z'_S(t)$ の(定常)**共分散**(covariance)関数 $C(\tau) = \mathrm{E}[Z'_S(t+\tau)Z'_S(t)] - \mathrm{E}^2[Z'_S(t)]$ に依存するものである．特に，$\bar{U}(l,\, T)$ は，過程 $Z'_S(t)$ が時間間隔 $T$ 内に水準 $l \geq 0$ を下方から突破する回数の長時間走行平均を表すものとする．$Z'_S(t)$ はガウス過程であるから，ライス(Rice)の古典的結果[24]

$$\bar{U}(l,\ T) = \frac{T}{2\pi\sigma} e^{-(l-m)^2/2\sigma^2} \left\{ -\frac{d^2C(\tau)}{d\tau^2} \bigg|_{\tau=0} \right\}^{\frac{1}{2}}$$

を応用することができる．ここで，$m$ と $\sigma^2$ は $Z's(t)$ の平均と分散である．［見本関数 $Z'_i(t)$ の連続性が上記の結果[24]にとって必要である．$Z_i(t)$ はこの性質を持たない．］$C(\tau)$ の第2階導関数を得るには，それが導関数の過程 $dZ's(t)/dt$ の共分散関数に等しいという結果[25]を利用することができる．5.5 節の議論を用いて次のことを確かめよ．

$$\lim_{t\to\infty} \Pr\left[\frac{dZ'_i(t)}{dt} = \alpha^{-1}\right] = \frac{\mathrm{E}(X)}{\mathrm{E}(X)(1+\beta/\alpha)+\mathrm{E}(I)}$$

$$\lim_{t\to\infty} \Pr\left[\frac{dZ'_i(t)}{dt} = -\beta^{-1}\right] = \frac{(\beta/\alpha)\mathrm{E}(X)}{\mathrm{E}(X)(1+\beta/\alpha)+\mathrm{E}(I)}$$

$$\lim_{t\to\infty} \Pr\left[\frac{dZ'_i(t)}{dt} = 0\right] = \frac{\mathrm{E}(I)}{\mathrm{E}(X)(1+\beta/\alpha)+\mathrm{E}(I)}$$

したがって，また次式をも確かめよ．

$$-\frac{d^2C}{d\tau^2}\bigg|_{\tau=0} = \lim_{t\to\infty} \mathrm{Var}\left(\frac{dZ's}{dt}\right)$$

$$= \frac{1+\alpha/\beta}{\alpha^2} \frac{N\mathrm{E}(X)}{\mathrm{E}(X)(1+\beta/\alpha)+\mathrm{E}(I)}$$

5.10 **図 P 5.10** に示す待ち行列ネットワークを考察しよう．システム内のジョブ数は一定の $N$ に維持されるものとする．すなわち，$IO_2$ プロセッサから1個の退去があると，即刻，1個の新しい到着が $IO_1$ 待ち行列に参入するものとする．CPU サービスに続いて，ジョブは，確率 $\alpha$ で $IO_2$ 待

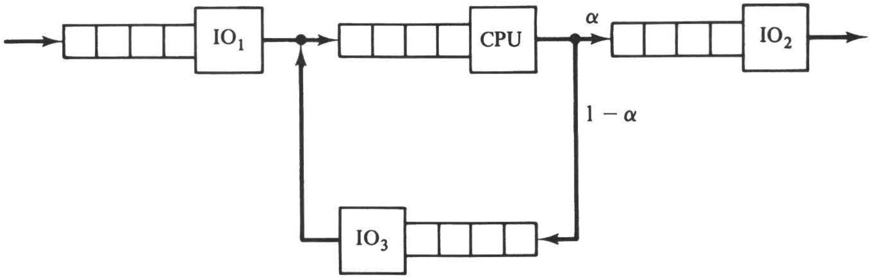



**図 P 5.10**　演習問題 5.10 のモデル

ち行列に加わり，$IO_3$ 待ち行列には確率 $1-\alpha$ で加わる．このシステムを **5.4** 節の方法を用いて解析せよ．($IO_1$ と $IO_2$ で表示される I/O 待ち行列は，ジョブの実行に付随する開始前の**搭載**(loading)と終了後の**卸下**(unloading)をモデル化する意味を持つ．)

**5.11**　(Kobayashi [26].) 本問題で演習問題 **5.10** と **5.4** 節のモデルを一般化することにする．特に，**図 P 5.11** の'開いた'システムを考えよう．そこでは，$\lambda$ はシステムへのポアソン到着率，$\lambda_i$ は $i$ 番目のプロセッサへの到着率であり，$\mu_i(n_i)$ は $i$ 番目のプロセッサにおける指数サービス時間の(状態依存型)パラメータであって，現に $i$ 番目のプロセッサ内にあるジョブと待ちつつあるジョブの数 $n_i$ の関数として与えられる．まず，次の式が成り立つことを確認せよ．

$$\lambda_j = \lambda\delta_{j0} + \sum_{i=0}^{N} \lambda_i p_{ij}$$

ただし，$i=0$ のとき $\delta_{i0}=1$，そうでないとき $\delta_{i0}=0$ であり，また，$p_{ij}$ は，$i$ 番目のプロセッサの出力が $j$ 番目のプロセッサへ進入する定常確率である．$p_i(n_i)$ は，$i$ 番目のプロセッサにおいてサービス中であるか待ちつつあるジョブの数 $n_i$ の平衡状態確率であるとし，$\mathbf{n}=(n_0, n_1, \ldots, n_N)$ はシステムの状態変数であるとしよう．

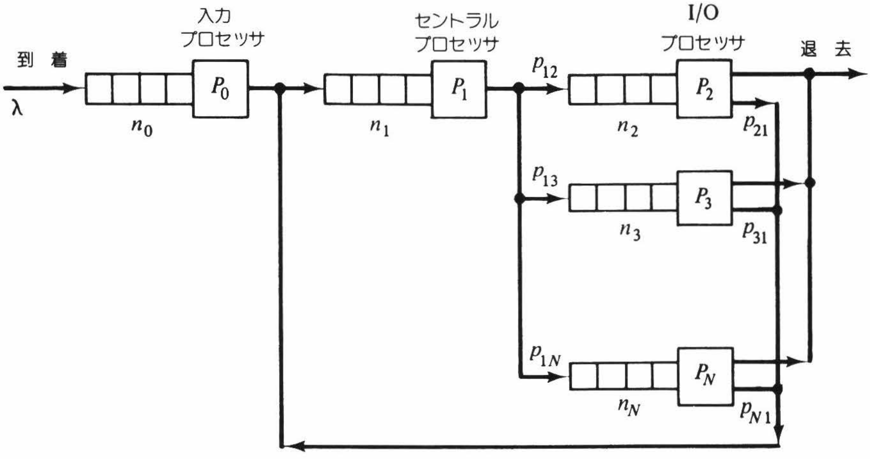



**図 P 5.11**　待ち行列ネットワークモデル

ポアソン型待ち行列の出力はそれ自身ポアソン過程である(演習問題4.8および4.11)という事実を用いて，分布 $\{p_i(n_i)\}$ は到着率 $\lambda_i$ とサービス時間パラメータ $\mu_i(n_i)$ を持つ独立のポアソン型待ち行列に対する結果によって決定されるということを確かめよ．したがって，4.3節から，次の式が成り立つ．

$$p_i(n_i) = p_i(0)\prod_{n=1}^{n_i} \frac{\lambda_i}{\mu_i(n)} \quad (n_i = 0, 1, 2, \ldots)$$

および

$$p(\mathbf{n}) = \prod_{i=0}^{N} p_i(n_i)$$

ただし，$p(\mathbf{n}) = \Pr[\mathbf{n} = (n_0, \ldots, n_N)]$．

$\mu_i(n_i) = \mu_i$ および $\rho_i = \lambda_i/\mu_i$ を仮定して，次式が成立することを示せ．

$$p(\mathbf{n}) = \prod_{i=0}^{N} (1-\rho_i) \prod_{i=0}^{N} \rho_i^{n_i}$$

任意のネットワークへのこのモデルの自然な拡張が Jackson [27] によって取扱われた．4.3節におけるように，到着率は状態依存型であってもよいので，システム動作について多くの変形を研究する手段を提供していることに注意するべきである．たとえば，5.4節の閉鎖型システムに対する結果はこの方法によって得られることを示す[26]方が却って容易である．

**5.12** 5.3節のモデルを参照すると，いわゆる CSCAN 規則は次のように SCAN 規則と異っている．要求にサービスを施すための走査は(定められた)1方向に向うときにだけ行われる．すなわち，完全な(側端から側端までの)‘サービス’走査に続いて，ヘッドは単位距離当り $a$ 時間の速さで開始位置に戻る．5.3節におけるその他の仮定と記号を用いて，CSCAN に対して次の式が成立することを示せ．

$$w(x) = \frac{a}{1-\lambda T} \quad (0 \le x \le 1)$$

この結果と SCAN および FSCAN に対する結果とを比較せよ．

**5.13** 5.2節のページング・ドラムに対して，退去要求(処理を完了した直後の要求)から見た所与のセクタ待ち行列内にある要求数の pmf$\{h_i\}$ の母関数

$H(z)$ を計算し，$H(z)$ が式(5.2.21)に等しいことを確かめよ．これを行うため，まず，式

$$g_n = g_0 f_n(T) + \sum_{i=1}^{n+1} g_i f_{n-i+1}(T) \quad (n = 1, 2, \ldots)$$

は，読取り/書込みヘッドが所与のセクタの始まりを通過するときに $n$ ($n \geq 1$) 個の要求を見出す平衡状態確率を定義することを確かめよ．次に，次式が成り立つことを示せ．

$$h_n = \sum_{i=1}^{n+1} \frac{g_i}{1-g_0} f_{n-i+1}\left(\frac{T}{N}\right)$$

**5.14** （端末メッセージ待ちタスク[28].）各端末からメッセージを並行して受信する互いに独立かつ統計的に同じ特性を持つ $n$ 個のタスクの集合を考えよう．各端末からは $\lambda$ の到着率で指数分布に従って文字が到来し，対応するタスクに割当てられた緩衝領域に記憶されるものとする．タスクがその緩衝領域から一つのメッセージを取出した瞬間から次にメッセージを取出すべく要求を出すまでの時間はパラメータ $\mu$ の指数分布に従う．各メッセージは末尾に EOL (end-of-line) 記号を伴う一連の記号から成る．$\{q_j\}_{j\geq 1}$ は EOL の位置 $j$ の pmf であり，$Q(z)$ はその母関数であるとする．もし，あるタスクがメッセージを取出す要求を出したときに，EOL 記号がその緩衝領域内になかったとしたならば，そのタスクは EOL 文字が到着するまで停止させられる．緩衝領域内に EOL 記号が少なくとも1個存在するときには，タスクは緩衝領域内の全内容を取出すものとする．

**A.** 確率変数 $x$ は，タスクの計算期間中緩衝領域内に蓄積される記号の数を表すものとする．次式が成り立つことを示せ．

$$\Pr[x = k] = (1-\sigma)\sigma^k \quad \left(k = 0, 1, 2, \ldots,\ \sigma = \frac{\lambda}{\lambda+\mu}\right)$$

**B.** 平均停止時間 $E(w)$ を求めるため，次のようにせよ．$j$ は EOL 記号の位置を表し，$x$ は最後に取出したとき以来蓄積された記号の数を表すものとする．このとき，次式が成り立つ．

$$E(w|j, x) = \begin{cases} 0 & (x \geq j) \\ \dfrac{j-x}{\lambda} & (x < j) \end{cases}$$

これを用いて，$E(w)=1/\lambda[m-\lambda/\mu(1-Q(\lambda/(\lambda+\mu)))]$ であることを示せ．ただし，$m=E(j)$ である．タスクが，到着と同時に停止する確率は $1-Q(\lambda/(\lambda+\mu))$ であることを示せ．

**C.** EOL 記号の位置 $j$ はパラメータ $q$ を持つ幾何分布 $q_j=q(1-q)^{j-1}$ $(j\geq 1)$ に従うものと仮定する．したがって，$E(j)=m=1/q$ となる．待ち時間の分布関数は次式によって定義されることを示せ．

$$dF(w)=\frac{\alpha\mu}{\alpha+\mu}e^{-\alpha w}dw \quad (w>0)$$

$$F(0)=\frac{\alpha}{\alpha+\mu}$$

ただし，$\alpha=q\lambda$ とする．

**D.** 全てのタスクは緩衝領域として十分な量の空間を割当てられていると仮定して，次式が成り立つことを示せ．

$$\Pr[k\text{ 個のタスクが停止する}]=\binom{m}{k}(1-\beta)^k\beta^{n-k}$$
$$(k=0,1,\ldots,n)$$

ただし，

$$\beta=\frac{1/\mu}{E(w)+1/\mu}$$

は，タスクが待機しながら過す時間の割合である．

## 参考文献


1. Fife, D. W., and J. L. Smith, "Transmission capacity of disk storage systems with concurrent arm positioning." *IEEE Trans. EC-14*, 4 (Aug. 1965), 575–582.
2. Denning, P. J., "Effects of scheduling on file memory operations." *Proc. AFIPS Conf. 30* 1967 Spring Joint Computer Conference 9–21.
3. Abate J., H. Dubner, and S. B. Weinberg, "Queueing analysis of the IBM 2314 disk storage facility." *J. ACM 15*, 4 (Oct. 1968), 577–589.
4. Seaman, P. H., R. A. Laird, and T. L. Wilson, "On teleprocessing system design. Pt IV: Analysis of auxiliary-storage activity." *IBM Sys. J. 5*, 3 (1966), 158–170.
5. Frank, H., "Analysis and optimization of disk storage devices for time-sharing systems." *J. ACM 16*, 4 (Oct. 1969), 602–620.

6. TEOREY, T. J., and T. B. PINKERTON, "A comparative analysis of disk scheduling policies," in *Proc. 3rd Symp. Op. Sys. Princ.* (Oct. 1971), 114–121.

7. KNUTH, D. E., *The Art of Computer Programming* (Vol. I). Addison-Wesley, Reading, Mass. (1968).

8. ABATE, J., and H. DUBNER, "Optimizing the performance of a drum-like storage." *IEEE Trans. Comp. C-18*, 11 (Nov. 1969), 992–996.

9. COFFMAN, E. G., JR., "Analysis of a drum input/output queue under scheduled operation in a paged computer system." *J. ACM 16*, 1 (Jan. 1969), 73–90. [Erratum: *J. ACM 16*, 4 (Oct. 1969), 646.]

10. WEINGARTEN, A., "The Eschenbach drum scheme." *Comm. ACM 9*, 7 (July 1966), 509–512.

11. BURGE, W. H., and A. G. KONHEIM, "An accessing model." *J. ACM 18*, 3 (July 1971), 400–404.

12. FELLER, W., *An Introduction to Probability Theory and Its Applications* (Vol. II). Wiley, New York (1966).

13. COX, D. R., *Renewal Theory*. Methuen, London (1962).

14. WALLACE, V. L., and R. S. ROSENBERG, "Markovian models and numerical analysis of computer system behavior." *Proc. AFIPS SJCC 28* (1966), 141–148.

15. IRANI, K. B., and V. L. WALLACE, "On network linguistics and the conversational design of queueing networks." *J. ACM 18*, 4 (Oct. 1971), 616–629.

16. MITRANI, I., "Non-priority multiprogramming systems under heavy demand conditions from a customer's point of view." Technical report No. 11, *Computing Laboratory*, University of Newcastle upon Tyne, Newcastle upon Tyne, England (Aug. 1970).

17. GORDON, W. J., and G. F. NEWELL, "Closed queueing systems with exponential servers." *Opns. Res. 15*, 2 (Apr. 1967), 254–265.

18. BUZEN, J., *Queueing Network Models of Multiprogramming*. Ph.D. thesis, Division of Engineering and Applied Physics, Harvard Univ., Cambridge, Mass. (June 1971).

19. LEWIS, P. A. W., and G. S. SHEDLER, "A cyclic-queue model of system overhead in multiprogrammed computer systems." *J. ACM 18*, 2 (Apr. 1971), 199–220.

20. GAVER, D. P., JR., and P. A. W. LEWIS, "Probability models for buffer storage allocation problems." *J. ACM 18*, 2 (Apr. 1971), 186–198.

21. DELGAVIS, I., and G. DAVISON, "Storage requirements for a data exchange." *IBM Sys. J. 3*, 1 (1964), 2–31.

22. Coffman, E. G., Jr., L. A. Klimko, and B. Ryan, "Analysis of scanning policies for reducing disk seek times." *SIAM J. Computing* (to appear).

23. Coffman, E. G., Jr., and A. C. McKeller, "On the motion of an unbounded, Markov queue in random-access storage." *IEEE Trans. Comp. C-17*, 6 (June 1968), 600–603.

24. Cramer, H., and M. R. Leadbetter, *Stationary and Related Stochastic Processes*. Wiley, New York (1967).

25. Yaglom, A. M., *An Introduction to the Theory of Stationary Random Functions*. Prentice-Hall, Englewood Cliffs, N.J. (1962).

26. Kobayashi, H., "Some Recent Progress in Analytic Studies of System Performance," *IBM Research Report*. T. J. Watson Research Center, Yorktown Heights, N.Y. (1972).

27. Jackson, J., "Job shop-like queueing systems." *Manag. Sci. 10*, 1 (1963), 131–142.

28. Denning, P. J., "A model for console behavior in multiuser computers." *Comm. ACM 11*, 9 (Sept. 1968), 605–612.

# 6 ページング・システムにおける記憶領域割当て

## 6.1 は じ め に

**記憶領域管理**(memory management)——または**記憶領域割当て**(storage allocation)——問題の定式化は，スケジューリング問題のそれと比較すると，極く最近になってから行われ出したといえる．研究されてきた話題には，最適必要時ページング・アルゴリズム[1]，記憶階層の効率的なシミュレーション設計のためのページング・アルゴリズム[2]，ある種のページング・アルゴリズムのマルコフ連鎖解析[3]，および拡張ページ化記憶システムの性能予測[4]がある．本章は，いくつかのさらに重要な結果を提供する．しかしながら，記憶領域割当て問題の範囲を，ここでは，相等しい大きさの区画またはブロックを単位として記憶領域を割当てるページング・システムに限定するように選択を行った．一般のシステムにおける記憶領域割当ての数学的モデルからは成果がほとんどない．というのは，最も単純な問題でさえもかなり難しいという傾向があるからである．一般論については Knuth [5] を参照するとよい．Knuth の扱いに関係あるもっと最近の結果は[6, 7]に見出すことができる．記憶領域割当てに関して多重プログラミング・システムに固有の問題は第7章に譲る．

ここでは，主記憶と補助記憶の二つのレベルから成るシステムに限って注目することにする．プログラムの**名前空間**(name space) $N$ は，それぞれが $c$ 個の連続する番地から成る**ページ**(page)に分割され，一方，システムの**主記憶空間**(main memory space) $M$ は，それぞれが $c$ 個の連続する**位置の番地**(location address)から成る**ページ枠**(page frame)に分割されているものと仮定する．本章を通じて，参照されるページ(またはページ枠)の番号だけに関心があり，ページ内の個々の語には関係しないので，$N = \{1, 2, ..., n\}$ をページの集合，$M = \{1, 2, ..., m\}$ をページ枠の集合として，$1 \leq m \leq n$ を仮定する．

ある整数がページを表すのか，あるいはページ枠を表すのかは常に明示されるから，この記法のあいまいさから混乱が生じることはないはずである．

各時刻 $t$ において，次のような**ページ写像**(page map) $f_t$: $N \to M \cup \{0\}$ が存在する．

$$f_t(x) = \begin{cases} y & \text{(ページ } x \text{ が時刻 } t \text{ にページ枠 } y \text{ に滞留しているとき)} \\ 0 & \text{(ページ } x \text{ が時刻 } t \text{ に } M \text{ 内にないとき)} \end{cases}$$

物理的には，ページ写像はプロセッサと記憶装置の間に介在する**番地変換**(address-translation)ハードウェアの一部分である．プロセッサが番地 $\alpha$ を生成すると，このハードウェアは記憶位置の番地 $\beta = f_t(x)c + w (0 \leq w < c)$ を算出する．ただし，$x$ は式 $\alpha = xc + w$ から定められる．(通例として) $c$ が2の冪になっているという性質を利用して，この計算を非常に効率的に行うようにハードウェアは構成されている．もし $f_t(x) = 0$ であるならば，適当なシステム・タスクに信号を送って，主記憶内に**欠落しているページ**(missing page)を補助記憶装置から取出し，それを主記憶装置内に格納し，写像 $f_t$ を更新して，再び**参照**(reference)を試みるように**ページ落ち割込み**(page fault interrupt)をハードウエアが惹き起こす．

1語に到達するのに要する平均時間は，それを含むページが $M$ 内にあるとき $\Delta_M$ であるとし，1ページを補助記憶および主記憶装置間で転送するのに要する平均時間は $\Delta_A$ であるとすると，速度比 $\Delta = \Delta_A / \Delta_M$ は重要なシステム・パラメータとなる．多くの現存システムは $\Delta \leq 10^4$ であるという仮定の上に立って作られているにもかかわらず，10または100の位の $\Delta$ を持つ新しい記憶装置構成が普通になってきていることを念頭に置くことが大切である．それゆえ，モデル化においてはモデルの鍵となるような $\Delta$ の大きさについての仮定は何も設けないように心掛けなければならない．

与えられた入力に対してプログラムが示すページングに関する振舞は，その(ページ)**参照列**(reference string)によって，機械に独立な言葉で記述することができる．参照列とは次のような系列である．

$$\omega = r_1 r_2 \ldots r_t \ldots \quad (r_t \in N,\ t \geq 1)$$

離散的時間パラメータ $t = 1, 2, 3, \ldots$ は，‘プロセス時間(処理時間)’ または ‘実行時間’ の流れの上での時刻を表現し，$r_t = i$ は，$t$ 番目の参照でページ $i$ が参照されることを意味する．長さ $T$ の全ての参照列の集合を記号 $N^T$ で表そう．また，$N^+$ は，空である列は含まないような $N$ の上の全ての有限長の列

の集合を表す.

参照列 $r_1r_2 \ldots r_t \ldots$ には, $a_t$ が $r_t$ の参照される時刻であるような実時間上での時刻の系列 $a_1a_2 \ldots a_t \ldots$ が対応する. プロセス時間上の時刻 $t$ と $t+1$ の間に経過する実時間は, 近似的に次式のように表される.

$$(6.1.1) \qquad a_{t+1} - a_t = \begin{cases} \Delta_M (r_{t+1} \text{ が記憶領域内にあるとき}) \\ \Delta_M + \Delta_A (\text{そうでないとき}) \end{cases}$$

後に, 参照列 $\omega$ の処理中に遭遇するページ落ち[†]の回数を $\omega$ の長さで正規化したもので定義される**ページ落ち率**(fault rate) $F(\omega)$ を導入する. もしページ取込み待ちと実行の時間が重複しないならば, プロセス時間上の時刻 $t$ と $t+1$ の間に経過する実時間の平均は近似的に次のようになる.

$$(6.1.2) \qquad \begin{aligned} \mathrm{E}[a_{t+1} - a_t] &= \Delta_M(1 - F(\omega)) + (\Delta_M + \Delta_A)F(\omega) \\ &= \Delta_M(1 + \Delta F(\omega)) \end{aligned}$$

こうして, 一般に近似的ではあるけれども, プロセス時間で表現される結果を実時間で観測されるプログラムの振舞に関係付ける簡単な手段が得られたことになる. 簡単であるのは望ましいということもさることながら, このことは, プロセス時間でプログラムの振舞を特徴付けるように問題を限定するのを正当化する.

長さ $T$ の参照列 $\omega$ をランダム系列と見るならば, $\omega$ の出現確率が $p(\omega)$ で表されるような $N^T$ の上の確率塊関数 $p(\omega)$ の存在を仮定することになる. そのような確率分布の存在を基にして構成される確率モデルは種々のページング・アルゴリズムの性能評価において最大の力を発揮することになろう.

## 6.2 ページング・アルゴリズム

ページング・アルゴリズムは, 次の三つの副方策を実現するものである. 1) **取込み時選択方策**(fetch policy)は, どのページが, そしてそれが何時, 主記憶に取込まれるかを決定し, 2) **格納位置選択方策**(placement policy)は, 取込まれたページを格納するために使用可能なページ枠を選択する. また, 3) **排除ページ選択方策**(replacement policy)は, 新入のページのために使用可能な空間を作るため主記憶から一つまたは複数のどのページを取除くかを決定する.

ページ枠は論理的に互いに等しいものと仮定しているから, 格納位置選択方

---

†**訳注**　必要時に目的のページが主記憶内に存在していないという事象をページ落ちという.

策がページング・アルゴリズムの振舞に影響を及ぼすことはあり得ない．（格納位置選択方策は，$k$ 個のページ枠を得るためには排除ページ選択方策を $k$ 回呼び出し，それらを得られた順序に従って割当てる．）したがって，ページング・アルゴリズムの振舞は取込み時選択と排除ページ選択の方策にだけ依存する．下記で形式的に定義される**必要時ページング**(demand paging)の下では必要時取込み方策が用いられ，そこでは，欠落ページだけが取込まれ，ページ排除は記憶領域が一杯のときにだけ起こる．それゆえ，必要時ページング・アルゴリズムの振舞は，その排除ページ選択方策にだけ依存する．

特に断らない限り，ページの大きさ $c$，記憶領域の大きさ $m$，速度比 $\Delta$，およびプログラムの語がページ間に分布する仕方は固定するものと仮定する．さらに，ただ一つのプログラム——与えられたページ集合 $N$ を伴うもの——が主記憶を占有し，それはページ落ちの場合を除いて割込まれることはない．これらの仮定の下で，システムの振舞に影響を及ぼすことのできる改善はページング・アルゴリズムを通してだけである．

もっと形式的な記述の仕方で先へ進む前に，本章を通じて用いられるいくつかの記法上の規約を導入しておく．集合は常に大文字(イタリック体)で表す．集合の要素は，普通，小文字(イタリック体)で表す．式の左辺が集合を表すとき，右辺に現われる ＋ (プラス)と － (マイナス)の演算記号は，集合の加算と減算であると解釈するべきである．また，式 $S=S'+x-y$ は，$S=S'+\{x\}-\{y\}$ の意味に取らなければならない．これらの規約によって，この記法は，関数 $R(\cdot)$ が時折出現することを除いて，あいまいになることはなかろう．$R(\cdot)$ は**排除ページ**(replacement page)($N$ の要素)を表しているので，たとえば，$S=S'+x-R(\cdot)$ のような式は，$S=S'+\{x\}-\{R(\cdot)\}$ を意味するものと取るべきである．上記の規約は，本章に現れる式の前後関係から直接に示唆されるので，何も困難は生じないはずである．

抽象的には，**ページング・アルゴリズム**(paging algorithm) $A$ は，参照列 $\omega=r_1r_2...r_t...$ を処理して，それに応じて**記憶状態**(memory state)の系列 $S_0S_1...S_t...$——ただし，$S_0$ は与えられた初期記憶状態である——を生成する機構である．各記憶状態 $S_t$ は，時刻 $t$ において $M$ 内に存在する $N$ から取られたページの集合であり，次の条件を満足する．

(6.2.1) $$S_t\subseteq N,\quad |S_t|\leq m,\quad r_t\in S_t\quad (t>0)$$

ここで，$|S|$ は集合 $S$ の要素の個数を表す．便宜上，初期記憶状態は空である，すなわち，$S_0=\phi$ であると仮定する．けれども，これは，取り扱いの一

般性に影響を及ぼすものではない．さらに，$S_t$ と $S_{t-1}$ は次式によって互いに関係づけられている．

$$S_t = S_{t-1} + X_t - Y_t \tag{6.2.2}$$

ただし，$X_t \subseteq N - S_{t-1}$ は時刻 $t$ において取込まれるページの集合であり，$Y_t \subseteq S_{t-1}$ は同時刻に排除されるページの集合である．

各時点 $t$ における $X_t$ と $Y_t$ を決定するため，ページング・アルゴリズム $A$ は内部記録を維持しなければならない．この記録は，初期制御状態を $q_0$ とする**制御状態**(control state)の集合 $Q$ によって表現される．アルゴリズムの**態勢表示**(configuration)は，$|S| \leq m$ および $q \in Q$ であるような対$(S, q)$である．アルゴリズムに結び付いて，次式のような**遷移関数**(transition function)$g_A$ がある．

$$g_A(S, q, x) = (S', q') \quad (x \in S') \tag{6.2.3}$$

ただし，$(S, q)$は現在の態勢表示，$(S', q')$は次の態勢表示，$x$ は遷移を惹き起こす参照である．特に，記憶状態の系列 $S_0S_1 \ldots S_t \ldots$ は，次式によって生成される態勢表示の系列$(S_0, q_0)(S_1, q_1)\ldots(S_t, q_t)\ldots$ に含まれる．

$$(S_t, q_t) = g_A(S_{t-1}, q_{t-1}, r_t) \quad (t \geq 1) \tag{6.2.4}$$

本章での議論では多くの場合，$A$ の制御状態に露には触れないで済ますはずである．

ページング・アルゴリズム $A$ は，与えられた $m>0$ に対して遷移関数が $g_A(S, q, x) = (S', q')$ ならば次式を満足するという性質を持つとき，**必要時ページング・アルゴリズム**(demand paging algorithm)であるという．

$$S' = \begin{cases} S & (x \in S \text{ のとき}) \\ S + x & (x \notin S,\ |S| < m \text{ のとき}) \\ S + x - y(\text{ある } y \in S \text{ に対して}) & (x \notin S,\ |S| = m \text{ のとき}) \end{cases} \tag{6.2.5}$$

したがって，必要時ページング・アルゴリズムの下では，$0 \leq |Y_t| \leq |X_t| \leq 1$ が成り立つ．排除するべきページ $y$ の選択権は，無論のこと，アルゴリズム $A$ の特権として認められている．態勢表示が$(S, q)$であるときに，$A$ が $S$ から排除するページを表すために記号 $R(S, q, x)$ を用いる．したがって，式(6.2.5)で，$y = R(S, q, x)$ が成り立つ．

参照列 $r_1r_2\ldots r_t\ldots$ において，ページ $x$ の時刻 $t$ での**前向き距離**(forward dis-

tance)$d_t(x)$ は，時刻 $t$ 後 $x$ の最初の参照までの距離である．すなわち，

$$(6.2.6)\quad d_t(x) = \begin{cases} k & (r_{t+k} \text{ が } r_{t+1}, r_{t+2}, \ldots \text{ における } x \text{ の最初の出現であるとき}) \\ \infty & (r_{t+1}, r_{t+2}, \ldots \text{ に } x \text{ が現れないとき}) \end{cases}$$

同様に，**後向き距離**(backward distance)$b_t(x)$ は，$x$ の最後の参照($x$ の最終参照ともいう)までの距離である．すなわち，

$$(6.2.7)\quad b_t(x) = \begin{cases} k & (r_{t-k} \text{ が } r_1r_2\ldots r_t \text{ における } x \text{ の最後の出現であるとき} \\ \infty & (r_1r_2\ldots r_t \text{ に } x \text{ が現れないとき}) \end{cases}$$

**例**：下記のものは，上記で導入した形式化を用いて記述された普通の必要時ページング・アルゴリズムの例である．$S=S_t$ は現在の記憶状態を，$x=r_{t+1}$ は次の参照を，$S'=S_{t+1}$ は次の記憶状態をそれぞれ表すものとする．ここでは排除ページ選択規則だけを述べているので，$|S|=m$ かつ $x \notin S$ であるものと仮定する．前述のように，$R(S, q, x)$ は $S$ 内から排除されるページを表すから

$$S' = S + x - R(S, q, x)$$

である．下記の諸例は，排除ページとして，どの場合も $N$ のある部分集合の線形順序付けによって定まる最小または最大の要素を選ぶという意味でいずれも同じような構造を示している．排除ページ選択規則を手短かな文章で記述した後で，この共通構造をもっと形式的な記述で明らかにする．

1．LRU(**最終参照時刻順**，least recently used)．$S$ 内から排除されるページは最大の後向き距離を持つものである．したがって，

$R(S, q, x) = y$ であるのは，$b_t(y) = \max_{z \in S}[b_t(z)]$ であるとき，かつそのときに限る．

2．$B_0$(**ベラディの最適アルゴリズム**[8]，Belady's optimal algorithm)．排除されるページは最大の前向き距離を持つ．互角を解決する規則として辞書式順序付けを用いる．(互角は無限の前向き距離を持つページの間だけで起こるから，互角解決規則はページ落ち率に影響しないことに注意を払わなければならない．)したがって，

$R(S, q, x) = y$ であるのは，$y = \min_{z \in S^*}[z]$ であるとき，

かつそのときに限る．

ただし，$z \in S^*$ であるのは，$d_t(z) = \max_{u \in S}[d_t(u)]$ であるとき，かつそのときに限る．

3．LFU(**最小参照回数順**，least frequently used)．排除されるページは，使われること(参照回数)が最も少なかったものである．互角解決規則はLRUである．もっと形式的に述べるならば，$f_t(x)$ が $r_1r_2...r_t$ 内の $x$ の参照回数を表すものとすると，

$R(S, q, x) = y$ であるのは，$b_t(y) = \max_{z \in S^*}[b_t(z)]$ であるとき，かつそのときに限る．

ただし，$z \in S^*$ であるのは，$f_t(z) = \min_{u \in S}[f_t(u)]$ であるとき，かつそのときに限る．

4．FIFO(**先着順**，first-in-first-out)．排除されるページは最も長い間続けて記憶領域内に滞在しているものである．$g_t(z) = i$ は，$S_j - S_{j-1} = r_j = z$ が成り立ち，しかも，$t$ より大でないような整数 $j$ の中で最大のもの† であると定義すると，

$R(S, q, x) = y$ であるのは，$g_t(y) = \min_{z \in S}[g_t(z)]$ であるとき，かつそのときに限る．

5．LIFO(**後着順**，last-in-first-out)．排除されるページは最も短い間続けて記憶領域内に滞在したものである．すなわち，

$R(S, q, x) = y$ であるのは，$g_t(y) = \max_{z \in S}[g_t(z)]$ であるとき，かつそのときに限る．

$R(S, q, x)$ が $r_{t+2}r_{t+3}...$ に無関係であるとき，$A$ は**非先見性**(nonlookahead)のアルゴリズムであるという．LRU, LFU, LIFO, および FIFO といったアルゴリズムは非先見性アルゴリズムの例であるが，$B_0$ は**先見性**アルゴリズムである．非先見性アルゴリズムは，参照列の知識を前もって持たなくても作動するという意味で実現可能である．明らかなことであるが，ページ参照列をランダム系列としてモデル化する確率モデルは，これらの性能を解析するための自然な接近法を齎す．(6.6節ではそのようなモデルの解析に立戻ることになる．)$B_0$ アルゴリズムは，実現不可能であるにもかかわらず，実現可能アル

†**訳注**　すなわち，時刻 $t$ から過去を振返ってみると，$z$ に対する最後の取込み時刻が $i$ である．

ゴリズムの性能を比較するための有用な比較標準を提供する． $B_0$ は各参照列に対するページ落ちの回数を最小にするものであるということの証明は次節で与えられる．

本章で採用した基本モデルは構造的に第2章と第3章のモデルからはそれほど隔たっていないということが看取されるのは少々面白い．実行系列に対応してページ参照列があり，システム状態に対応して態勢表示がある．しかし，ページ参照の先行後続関係には全く関心がないという点に大きな相違があるのは勿論のことである．

## 6.3 最適ページング・アルゴリズム

### 6.3.1 費用関数

式(6.2.2)から，記憶状態 $S_t$ と $S_{t-1}$ は $S_t = S_{t-1} + X_t - Y_t$ によって関係づけられている．ただし，$X_t \cap Y_t = \phi$ である．大きさ $m$ の記憶領域において参照列 $\omega = r_1 r_2 \dots r_T$ の上で作動するページング・アルゴリズム $A$ によって生じる**費用**(cost)を，$C(A, m, \omega)$ で表す．一般に，費用は，系列 $\{X_t\}$ および $\{Y_t\}$ のある関数である．というのは，これらの系列は補助記憶とのやりとりを体現しているからである．実際上は，$\{Y_t\}$ による費用成分は，ある一定の比率を $\{X_t\}$ による成分に掛けたものに等しいという傾向がある．そこで，$\{Y_t\}$ を無視することにより $C(A, m, \omega)$ の定義を簡単化することができる．さて，$h(k)$ は，補助記憶に対する1回の処置で $k$ ページを一纏めにして転送するのに要する費用を表すものと仮定する．簡単のため，1ページの転送は単位費用を要し，また

$$h(0) = 0, \quad h(k) \geq h(1) = 1 \tag{6.3.1}$$

が成り立つと仮定する．

実用的に興味のあるシステムにおいて，$h(k)$ は，$k$ に関して単調非減少であることも特に注意しておく．$k$ に関する $h(k)$ の増加率は，すぐ後で分るように，必要時ページング・アルゴリズムの最適性に影響を持つ．費用を次のように定義することにしよう．

$$C(A, m, \omega) = \sum_{t=1}^{T} h(|X_t|) \tag{6.3.2}$$

$A$ が必要時ページング・アルゴリズムであるならば，$1 \leq t \leq T$ に対して $|X_t| \leq 1$ であり，費用は単に

$$C(A,\ m,\ \omega) = \sum_{t=1}^{T} |X_t| \tag{6.3.3}$$

となる．確率分布 $p(\omega)$ が $N^+$ の上で定義されているとき，**平均費用**(expected cost)を次のように定義する．

$$C(A,\ m) = \sum_{\omega \in N^+} p(\omega) C(A,\ m,\ \omega) \tag{6.3.4}$$

全ての $m$ と $\omega$ に対して $C(A,\ m,\ \omega)$ を最小にするならば，ページング・アルゴリズムは最適である．しかし，非先見性アルゴリズムを扱うときは，$N^+$ の上の分布 $p(\omega)$ が与えられたと仮定して，$A$ が $C(A,\ m)$ を最小にするならば，$A$ は $p(\omega)$ に関して最適であると定義しよう．

1ページの転送は $T_w + T_t$ 時間を要する．ただし，$T_w$ は待ち時間(たとえば，待ち行列における遅れ，回転媒体を用いた補助記憶装置における回転待ち時間)であり，$T_t$ はページ送出時間である．$k$ ページの転送は，補助記憶装置が電子式選択方式(たとえば，磁心記憶装置)を採っているならば，$k(T_w + T_t)$ 時間を要するであろうし，補助記憶装置が回転式選択方式(たとえば，ディスクまたはドラム)を採っているならば，$T'_w + kT'_t$ 時間掛かるであろう．普通は $T'_w > T_w$ である．前者の場合には $h(k) = k$ となり，後者の場合には $h(k) < k$ となる．たとえば，ある $0 < a < 1$ に対して $h(k) = 1 + a(k-1)$ (ただし，$k \geq 1$)である．すぐ後で与えられる定理 6.1 は，$h(k) \geq k$ であるならば，最適な(すなわち，費用を最小にする)アルゴリズムのクラスは必要時ページング・アルゴリズムを含まなければならないということを示している．現用のシステム(磁心-ドラム・システム)では $h(k) < k$ である場合が往々にしてあるにもかかわらず，最適必要時アルゴリズムの存在を決定するための $h$ についての簡単な条件というものはない．この条件は，一般に，参照列に対する確率の割当てと $k - h(k)$ の大きさに依存するであろう．

$h(k) = k$ である場合は，半導体主記憶装置と磁心を用いた**後背記憶装置**(backing store)の使用(すなわち，**キャッシュ**(cache)と**隷属記憶**(slave memory))が増加するに従って優位を占めるようになるであろう†．この場合，定理 6.1 は，システム設計者が最適または**近最適**(near-optimal)なアルゴリズムを探し求めるとき，必要時アルゴリズムだけに注目すればよいということを明らかにしている．この理由で，また，[$h(k) < k$ であろうとなかろうと]必要時

---

†**訳注**　原文ではこうであるが，記憶階層の構成法には計算機の発達段階によって変遷があり，現在の大型計算機では，高速半導体緩衝記憶，半導体主記憶および磁気ディスクの3階層，またはさらに中速大容量半導体緩衝記憶を付加した4階層の構成法が採られている．

ページングで実現されている大多数のシステムの範囲内で現在得られている結果が考えられたという理由で，ここでは，定理**6.1**に従う必要時ページング・アルゴリズムに注意を限定することにし，**アルゴリズム**という言葉で**必要時ページング・アルゴリズム**を指すことにする．

**定理** 6.1

$h(k) \geq k$ であると仮定する．任意に与えられたページング・アルゴリズム $A$ に対して，ある必要時ページング・アルゴリズム $A'$ が存在し，次式

$$C(A', m, \omega) \leq C(A, m, \omega)$$

が全ての $m$ と $\omega$ に対して成り立つ．

**証明** 証明は，$A$(ある種のプリページング†—prepaging—を行うかも知れない)が，参照されること無しに後になって排除されるようなページを記憶領域内に持ち込むかも知れないという事態を取除こうという考え方に基づく．必要時アルゴリズム $A'$ は，このことを，$A$ が下すであろう決定を制御状態の中に憶え込み，それをページ落ちが起こるまで引延ばすことによって行う．$A$ は，記憶状態の系列 $\{S_t\}$——これは，関係 $S_t = S_{t-1} + X_t - Y_t (X_t \cap Y_t = \phi)$ に従う系列 $\{(X_t, Y_t)\}$ に等価である——を生成するものと仮定する．同様に，$A'$ は $\{S'_t\}$ を生成する．アルゴリズム $A'$ は $A$ をシミュレートし，$A$ によっては行われるが $A'$ によっては未だ行われないページ格納の記録を集合 $P_t$(繰延べ格納ページの集合)に残し，$A$ によっては行われるが $A'$ によっては未だ行われないページ排除の記録を集合 $R_t$(繰延べ排除ページの集合)に残す．$S_t$ と $S'_t$ との間の関係は次の通りである．

$$S'_t = S_t + R_t - P_t$$

時刻 $t$ での繰延べ格納ページは，時刻 $t-1$ で繰延べ格納ページであったか，あるいは時刻 $t$ に $A$ によって取込まれたページではあるが，$A'$ によっては時刻 $t$ までに取込まれて記憶領域内に存在しているページではないようなものである．したがって，

$$P_t = (P_{t-1} + X_t - Y_t) - (S'_{t-1} + r_t)$$

である．新しい記憶状態 $S'_t$ は，$S'_t = S'_{t-1} + X'_t - Y'_t$ として定義される．

†**訳注** 必要を予想して，あるいは単に機械的に前もってページを取込むこと．

ただし，$X'_t$ は $r_t \in S'_{t-1}$ ならば空であり，そうでなければ $r_t$ に等しい．$Y'_t$ は，$X'_t$ が空であるかまたは $|S'_{t-1}| < m$ であれば空であり，そうでなければある $y \in R_{t-1} + Y_t$ に対して $Y'_t = \{y\}$ である．($R_{t-1} + Y_t$ は1ページよりも多くのページを含み得るので，$A'$ は一意ではないことに注意しよう.)時刻 $t$ での繰延べ排除ページは，時刻 $t-1$ で繰延べ排除ページであったか，または時刻 $t$ に $A$ によって排除されたページであり，しかも $A'$ の下では時刻 $t$ に記憶領域内に存在するものに限る．したがって，

$$R_t = (R_{t-1} + Y_t - X_t) \cap S'_t$$

である．構成の仕方から，$A'$ によって格納された全てのページは $A$ によっても格納されていなければならない．このことは，$\sum_t |X_t| \geq \sum_t |X'_t|$ を意味する．したがって，

$$C(A, m, \omega) = \sum_t h(|X_t|) \geq \sum_t |X_t| \geq \sum_t |X'_t| = C(A', m, \omega)$$

である．ここで最初の不等号に対しては $h(k) \geq k$ を用い，最後の等号に対しては式(6.3.3)を用いている．■

### 6.3.2 最適排除ページ選択方策

必要時ページングに対して提案されてきた一つの排除ページ選択規準は，排除されるページの‘最良’の選択は次の参照までの時間の平均値が最大であるようなページ，すなわち，前向き距離の平均値が最大であるものを選ぶこと† であると主張している．その理論的根拠は，そうすることによって，ページを取戻すために起こるであろうページ落ちまでの時間間隔の平均値を最大にし，それにより，ページ落ち率，したがって，費用を最小にするというものである．もしも，完全な参照列の事前の知識を仮定してこの規準を実現することができるならば，本節において後に最適であることが証明される**決定性 $B_0$ アルゴリズム**(deterministic $B_0$ algorithm)が得られることになる．しかしながら，確率についての言明しか行うことができないとするならば，式(6.3.4)を最小にする排除ページの選択が最大平均前向き距離を持つページの選択とはならないような確率割当てを構成することが可能である．この事実は，次の幾分技巧的な例によって示される．

$N = \{1, 2, 3\}$, $m = 2$ とし，整数 $k$ の $j$ 個の連続出現を表す記号 $(k)^j$ を導

†**訳注**　この規準を**平均前向き距離規準**(expected-forward-distance criterion)という．

入する．可能な参照列は $1233(2)^j1$ と $12313(2)^j$ のただ二つであり，それぞれ確率 $1-p$ および $p$ で発生するものと仮定する．状態が $S=\{1,2\}$ で，ページ3が初めて参照されるときの排除ページの決定について調べることにする．参照列の残余を構成するのは $3(2)^j1$ であるのか，それとも $13(2)^j1$ であるのかということが分らないとすると，平均前向き距離は，それぞれ

$$d(1)=p+(j+2)(1-p)=1+(j+1)(1-p)$$
$$d(2)=3p+2(1-p)=2+p$$

である．$R(x)$ は，3番目の参照でページ $x$ が排除される場合の最小(総)排除回数の平均値であるとすると，

$$R(1)=3p+2(1-p)=2+p$$
$$R(2)=2p+2(1-p)=2$$

を得る．$p>0$ であるならば，ページ1が排除されるべきでないのは明らかである．しかし，$j>2p/(1-p)$ であるならば，$d(1)>d(2)$ となり，平均前向き距離規準はページ1を排除するよう助言をする誤りを犯すことになる．

**独立参照**(independent-reference)モデルは，6.6節で証明するように，前向き距離規準が最適アルゴリズムを生む確率構造であって，一般によく研究されている．さらに，この規準は，非常によい発見的方法を提供することが知られている．最後に述べたことに関係する実験的証拠は図6.3-1に要約されてい

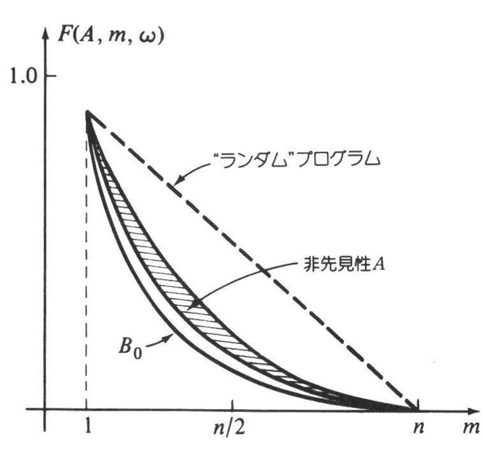



図6.3-1 ページ落ち率性能

る．この資料は文献[8, 10, 11]にあるデータから抽出したものである．これは，多数の‘典型的’なプログラムの実行から参照列の集合を抽出し，そこから選び出した典型的な(長い)参照列 $\omega$ に対して求められた記憶領域の大きさ $m$ に対するページ落ち率

$$F(A, m, \omega) = \frac{C(A, m, \omega)}{|\omega|} \tag{6.3.5}$$

を定量的に示したものである．非先見性アルゴリズムに対するページ落ち率曲線は図の斜線部分に入る傾向がある．典型的には，最良のアルゴリズム($B_0$)でさえ単位時間に許容し難い程多くのページ落ちを発生するような記憶領域の大きさの範囲が存在し，この領域の上限は典型的には $n/2$ の辺りにある．

もしページング・アルゴリズムが記憶領域の大きさの許容できる領域 $m>n/2$ で作動するならば，$F(A, m, \omega)$ と $F(B_0, m, \omega)$ は高々2，3パーセントしか違わないという傾向がある．これは，特にページ落ち率が受入れられる水準に保たれているときには，**プログラムの実行速度**(performance)**は $A$ に対するよりも $m$ に対して遙かに感受性がある**ということを示唆している．転じて，このことは，ページング・アルゴリズムの研究は最適なものを発見しようとすることよりも安定で実現が容易なアルゴリズムを見出し解析する技法の開拓にもっと力を注ぐべきであるということを示唆している．

図6.3–1は，プログラムの振舞のもう一つの重要な特色を示してもいる．この特色は，$F(A, m, \omega)$ 曲線が‘ランダム・プログラム’と記した直線の下方に横たわる傾向があると述べることができることである．各ページが各参照時に確率 $1/n$ で参照されるとき，プログラムはランダムである．この場合，全ての実現可能なアルゴリズムに対してページ落ち率の平均は $(n-m)/n$ となる．しかし，そのようなプログラムはかつて見たことがない．これまでに研究された非先見性アルゴリズムに対して $F(A, m, \omega)$ が直線 $(n-m)/n$ のかなり下方にあることは，プログラムがその参照を一部のページに密集させる傾向を持っていることを示している．**局所性**(locality)として知られるこの性質の研究と解明は第7章において行われる．

FIFOアルゴリズムは，図6.3–1に示される性能特性のためばかりでなく，実現するのが非常に容易であるという理由で興味がある．しかしながら，それは，ある一定の実際にあり得る条件の下で，奇妙で望ましくない振舞を露呈するという不利益がある[9]．すなわち，直観的には，任意の $A$ および $\omega$ に対して，$C(A, m, \omega)>C(A, m+1, \omega)$ が成り立つであろうこと，つまり，プログ

ラムにより多くの記憶空間を割当てれば，ページ落ち率は減少するであろうことを期待されるかも知れないが，実は，FIFO に対しては $C(A, m, \omega) < C(A, m+1, \omega)$ が成り立つような $\omega$ と $m$ が存在するのである [9]．後に，**スタック・アルゴリズム**(stack algorithm)はこのような異常な振舞をしないこと，さらに，LRU, $B_0$, LFU, および LIFO がスタック・アルゴリズムの例であることが明らかにされる．

FIFO の上述の変則的振舞の例として，$N = \{1, 2, 3, 4, 5\}$ とし，参照列 123412512345 を考える．$m = 3$ に対する記憶状態の系列は，FIFO 規準によって上下に並べられたページの列で下記の通りに示される．

$$\begin{array}{cccccccccccc}
1 & 1 & 1 & 2 & 3 & 4 & 1 & 1^* & 1^* & 2 & 5 & 5^* \\
 & 2 & 2 & 3 & 4 & 1 & 2 & 2 & 2 & 5 & 3 & 3 \\
 & & 3 & 4 & 1 & 2 & 5 & 5 & 5 & 3 & 4 & 4
\end{array}$$

ページ落ちによって更新され**ない**状態には星印が付されている．ところで，$m = 4$ に対しては，次の系列を得る．

$$\begin{array}{cccccccccccc}
1 & 1 & 1 & 1 & 1^* & 1^* & 2 & 3 & 4 & 5 & 1 & 2 \\
 & 2 & 2 & 2 & 2 & 2 & 3 & 4 & 5 & 1 & 2 & 3 \\
 & & 3 & 3 & 3 & 3 & 4 & 5 & 1 & 2 & 3 & 4 \\
 & & & 4 & 4 & 4 & 5 & 1 & 2 & 3 & 4 & 5
\end{array}$$

これから，$m = 3$ に対しては 9 回のページ落ちしか生じないのに，$m = 4$ に対しては 10 回ものページ落ちが起きていることが分る．さらに，より大きい記憶領域においてより小さい記憶領域におけるページング回数の 2 倍に達するページングが必要となるようにすることができるということを示す例も作れる(演習問題 6.7 を見よ)．

次に，$B_0$ の最適性を証明することにする．この結果は，文献 [2] で初めて証明された．また，別証が文献 [12] に示されている．ここでの証明は，後節のいくつかにおいて有用であることが分る**動的計画**(dynamic programming)法を用いる．

**定理 6.2**

アルゴリズム $B_0$ は，全ての $m$ と $\omega$ に対して $C(A, m, \omega)$ を最小にする．

**証明** $C_k(S, t)$ は，$S_t = S$ が与えられたとき，参照 $r_{t+1} \ldots r_{t+k}$ を処理するのに要する必要時ページングの下で達成可能な最小費用を表すものとする．もし $C_0(S, t) = 0$ と定義し，$r_{t+1} = x$ と置くならば，次のように書くことがで

きる．

$$(6.3.6)\qquad C_k(S,t)=\begin{cases}C_{k-1}(S,t+1) & (x\in S)\\ 1+\min_{y\in S}[C_{k-1}(S+x-y,t+1)] & (x\notin S)\end{cases}$$

この関係式は，動的計画問題における**最適性原理**(principle of optimality)に他ならないことに気付くであろう．定理の証明は，今や，$B_0$ に従うページングが式(6.3.6)によって特徴付けられることを示すことに帰着する．

$x<y$ は $x$ が時刻 $t$ において $y$ よりも大きい前向き距離を持つということを意味するように，<で，各時刻 $t$ におけるページ集合 $N$ の上の線形順序を表すものとする†．また，$\min[S]$ は，$S\subseteq N$ に含まれる $N$ の最小元を表す．$B_0$ が式(6.3.6)によって記述されることを示すためには，全ての $k\geq 0$ と $t\geq 1$ に対して，$B_0$ の選択規準に従って $s=\min[S]$ と置くならば，

$$(6.3.7)\qquad C_k(S+x-s,t)=\min_{y\in s}[C_k(S+x-y,t)]$$

が成り立つことを示さなければならない．式(6.3.7)を証明するには，$y$ と $z$ の双方を含む任意の $S$ に対して，$y<z$ であるならば，

$$(6.3.8)\qquad \Delta C_k=C_k(S-z,t)-C_k(S-y,t)\geq 0$$

が成り立つことを示せば十分である．実際，$\Delta C_k$ は0かまたは1であることを証明しよう．証明は $k$ についての帰納法による．$k=0$ に対してこの結果が真であることはいうまでもない．そこで，正である任意の $k$ よりも小さい全ての値に対して，これが成り立つものと仮定する．$r_{t+1}=x$ として式(6.3.6)を適用する際に，三通りに場合を区別して行うことができる．$y$ の前向き距離は $z$ のそれより大きい(すなわち，$y<z$)ことを仮定したので，$x=z$ であることは可能である．しかし，$x=y$ は不可能であることを注意しておく．

**場合 1.**　$x\in S-y-z$．式(6.3.6)から次式を得る．

$$\Delta C_k=C_{k-1}(S-z,t+1)-C_{k-1}(S-y,t+1)$$

これは，帰納法の仮定から，0か1である．

**場合 2.**　$x=z$．式(6.3.6)から

$$\Delta C_k=1+\min_{u\in S-z}[C_{k-1}(S-u,t+1)]-C_{k-1}(S-y,t+1)$$

† 前出と同じく，$d_t(x)=d_t(y)=\infty$ であるならば，$x$ が $y$ よりも辞書式の順序付けで小さいとき，かつそのときに限り $x<y$ である．

$u_0 = \min[S - z]$ と置くと†，明らかに，$u_0 = y$ かまたは $u_0 < y$ である．$k - 1$ に対して式(6.3.8)したがって式(6.3.7)が成り立つという帰納法の仮定から $C_{k-1}(S - u_0, t+1) = \min_{u \in S-z}[C_{k-1}(S-u, t+1)]$ であるから，$\Delta C_k$ は，前者の場合，1 となり，後者の場合は 0 か 1 となり，いずれにしても 0 または 1 である．

**場合 3．** $x \notin S$，式(6.3.6)から次式が成り立つ．

$$\Delta C_k = \min_{u \in S-z}[C_{k-1}(S - z + x - u, t+1)] - \min_{v \in S-y}[C_{k-1}(S - y + x - v, t+1)] \tag{6.3.9}$$

$S - z - y$ の最小元 $s = \min[S - z - y]$†† について三つの可能性がある．1) $s < y < z$ であるならば，式(6.3.9)は次式のようになる．

$$\Delta C_k = C_{k-1}((S + x - s) - z, t+1) - C_{k-1}((S + x - s) - y, t+1)$$

これは，帰納法仮定から，0 か 1 のどちらかである．2) $y < s < z$ であるならば，式(6.3.9)は次のようになる．

$$\Delta C_k = C_{k-1}(S - z + x - y, t+1) - C_{k-1}(S - y + x - s, t+1)$$

これもまた帰納法仮定により 0 か 1 のどちらかである．最後に，3) $y < z < s$ であるならば，次式が得られる．

$$\Delta C_k = C_{k-1}(S - z + x - y, t+1) - C_{k-1}(S - y + x - z, t+1) = 0$$

ゆえに，全ての場合について $\Delta C_k$ は 0 か 1 のどちらかであるといえる．■

次の二つの結果は，LRU と $B_0$ の振舞にさらに光を当てることになるという理由で興味深い．$\omega^R$ は $\omega$ の逆系列を表すものとする．そうすると，次式が成り立つ．

$$C(\mathrm{LRU}, m, \omega^R) = C(\mathrm{LRU}, m, \omega) \tag{6.3.10}$$

$$C(B_0, m, \omega^R) = C(B_0, m, \omega) \tag{6.3.11}$$

---

†**訳注** 原文では，'$u_0$ は $C_{k-1}(S - u_0, t+1)$ が最小値であるような $S - z$ 内のページである' となっている．しかし，これでは不都合である．というのは，このように定義すると，$d_{t+1}(y) = d_{t+1}(u_0) = \infty$ の場合に $y < u_0$ が成り立つ可能性があり，$u_0 \leq y$ とは言えなくなるからである．（しかし，原文のような $u_0$ の定義を用いても，結局は正しい結論が得られることを注意しておく．ただし，その場合，証明がもっと長くなる．）なお，続く文も，$u_0$ の定義の修正に合わせて，多少変更した．

††**訳注** 原文では '排除ページ $s = \min[S - z - y]$' となっているが，これは，誤った表現（$s$ は排除ページであるとは限らないから）であるので，本文のように訂正しておく．

式(6.3.10)を証明するため，$t_1, t_2, \ldots, t_i, \ldots, t_r$ は，$\omega$ において与えられたページ $x$ への参照が起こる時点の系列を表すものとする．$t_i$ と $t_{i+1}$ の間($t_i$ と $t_{i+1}$ は含まない)に参照される相異なるページの個数を $t_i$ と $t_{i+1}$ の間の**距離** $a_x(i)$ と定義する．さて，$a_x(i)=k$ と仮定する．このとき，$t_{t_{i+1}}$ は，$m>k$ であるとき，かつそのときに限り $m$ ページから成る記憶領域内にあることになる．そうすると，明らかに，$\omega$ におけるページ $x$ のページ落ち回数は，$\omega$ における距離 $a_x(i)$ の系列——これは $\omega^R$ におけるそれの逆である——だけに依存する．したがって，当然，$\omega$ における $x$ のページ落ち回数は，$\omega^R$ におけるそれに等しい．

式(6.3.11)を証明するため，$C(B_0, m, \omega^R)<C(B_0, m, \omega)$ を仮定しよう．さて，$\omega^R$ の上で $B_0$ は時点 $t_1, t_2, \ldots, t_p$ でページ落ちが起こるものとする．$\omega$ の上で次のように動作するアルゴリズム $A$ を定義する．$A$ は $\omega^R$ 上での $B_0$ における最後の記憶状態を最初の記憶状態とし，さらに，$B_0$ において $t_i(i=1, \ldots, p)$ に参照されるページを $A$ が参照すると，$A$ はその直後に記憶状態を $B_0$ における $t_i$ 直前の記憶状態に一致させる．すなわち，$\omega$ の上での $A$ に対する記憶状態系列は $\omega^R$ の上での $B_0$ のそれの逆になる．一般に，$A$ は非必要時アルゴリズムである．定理6.1により，$C(A', m, \omega)\leq C(A, m, \omega)$ が成り立つような必要時アルゴリズム $A'$ が存在する．また，定理6.2から $B_0$ は最適である．したがって，

$$C(B_0, m, \omega)\leq C(A', m, \omega)\leq C(A, m, \omega)=C(B_0, m, \omega^R)<C(B_0, m, \omega)$$

これは矛盾である．$\omega$ と $\omega^R$ の役割を逆にすると，$C(B_0, m, \omega^R)>C(B_0, m, \omega)$ もまた不可能であることが示される．ゆえに，式(6.3.11)が成立する．式(6.3.10)と(6.3.11)は共に文献[2]で証明された．その中で両式は，$\omega$ の順・逆方向走査を行って，$C(\mathrm{LRU}, m, \omega)$ と $C(B_0, m, \omega)$ の両方とも決定することができるアルゴリズムを構成するために使用されている．

## 6.4　スタック・アルゴリズム[2]

### 6.4.1　はじめに

$S_0=\phi$ と仮定して，アルゴリズム $A$ が必要時ページングの下で大きさ $m$ の記憶領域において参照列 $\omega$ を処理した結果得られる記憶状態を表すため，記号 $S(A, m, \omega)$ を導入する．すなわち，もし $\omega=r_1r_2\ldots r_t$ が $A$ の下で $S_0 S_1\ldots S_t$ を生成するならば，$S(A, m, \omega)=S_t$ である．前と同様，$A$ であると了解されるならば，簡単に $S(m, \omega)$ と書く．記号 $R(S, q, x)$ は，態勢表示

$(S, q)$において次の参照が $x$ であるときに $A$ が $S$ から排除するページを表す. $x \in S$ であるか, または $|S| < m$ であるときは, $R(S, q, x) = 0$ であると定義する.

アルゴリズム $A$ は, その記憶状態が全ての $m$ および $\omega$ に対して**包含特性**(inclusion property)

$$(6.4.1) \qquad S(m, \omega) \subseteq S(m+1, \omega)$$

を満足するとき, **スタック・アルゴリズム**(stack algorithm)であるという. すなわち, 記憶状態は, **入れ子**[†] **集合**(nested set)の族を形成する. 条件(6.4.1)は次の条件に等価である. 各 $\omega$ に対し, $N$ の順列 $\mathbf{s}(\omega) = [s_1(\omega), \ldots, s_n(\omega)]$ が存在して, 全ての $m$ に対し $S(m, \omega) = \{s_1(\omega), \ldots, s_m(\omega)\}$ が成り立つ. [等価性は, $S(0, \omega) = \phi$ および $\{s_i(\omega)\} = S(i, \omega) - S(i-1, \omega)$ と置くことによって得られる.] ベクトル $\mathbf{s}(\omega)$ は, **スタック・ベクトル**(stack vector), または単に**スタック**(stack)と呼ぶ. $i < j$ であるならば, $s_i(\omega)$ は, $s_j(\omega)$ よりもスタックにおいて位置が**上位**(higher)にあるといい, $s_1(\omega)$ はスタックの**先頭**(top)であるという. 初期スタックは空であり, もし $\omega$ が $k < n$ 個の相異なるページしか含まないならば, $\mathbf{s}(\omega) = [s_1(\omega), s_2(\omega), \ldots, s_k(\omega)]$ と定義する. したがって, 与えられたスタック・アルゴリズムに対して包含特性は, 各参照列 $r_1, \ldots, r_t$ に対してスタックの系列 $\mathbf{s}_1, \ldots, \mathbf{s}_t$ を次の条件が成り立つように構成し得るということを意味する. すなわち, $m$ の各値に対して, 単に各スタックの上位の $m$ ページを取ることによって, 記憶状態の系列が定められる.

**例**: LRU は, $S(\text{LRU}, m, \omega)$ が $\omega$ において最後に参照された $m$ ページをいつも含んでいて, したがって $S(\text{LRU}, m+1, \omega)$ の部分集合となるから, スタック・アルゴリズムである. さらに, LRU スタック $\mathbf{s}(\omega)$ は, 後向き距離の増加順にページを順序付けている. FIFO は, **表 6.4-1** で確かめられるように, スタック・アルゴリズムではない. $\omega$ の各参照ページの下の列は, 与えられた記憶領域の大きさに対する(格納順に並べられた)記憶内容を表現している. 明らかに, 最終記憶状態は包含特性を満足していない.

包含特性を持つアルゴリズムが有する主な利点の一つは, そのようなアルゴリズムの性能の研究において, 与えられた参照列のページ落ちに関する振舞を, 記憶領域の大きさ $m(1 \leq m \leq n)$ の全てに対して並列にかつ参照列の 1 回

[†]**訳注** 入れ子とは, 同じ形の箱などを大きさの順に幾つも重ねて中へ入れるようにしたものである.

表6.4-1　FIFOの例

| | $\omega$ | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 1 | 2 | 5 |
| $m=3$ | 1 | 2 | 3 | 4 | 1 | 2 | 5 |
| | — | 1 | 2 | 3 | 4 | 1 | 2 |
| | — | — | 1 | 2 | 3 | 4 | 1 |
| $m=4$ | 1 | 2 | 3 | 4 | 4 | 4 | 5 |
| | — | 1 | 2 | 3 | 3 | 3 | 4 |
| | — | — | 1 | 2 | 2 | 2 | 3 |
| | — | — | — | 1 | 1 | 1 | 2 |

の走査で効果的に計算することができるということである．このようにして，**記憶階層**(memory hierarchy)とページング・アルゴリズムの効率的なシミュレーション研究を行うことができる．スタック系列とそれに対応する費用関数 $C(A, m, \omega)$ [式(6.3.3)参照]を計算する効率的手続きは，本節の主要な結果である．しかしながら，その前に，包含特性が成り立つために必要なスタック遷移のいくつかの特性を記述し，スタック・アルゴリズムのもう少し構成的な定義を与えることにする．

ページ $x$ に対して，その**スタック距離**(stack distance) $D_x(\omega)$ は，スタック $\mathbf{s}(\omega)$ における $x$ の位置であると定義する．したがって，$s_k(\omega)=x$ であるならば，$D_x(\omega)=k$ である．もし $x$ が $\mathbf{s}(\omega)$ 内に現れないならば，$D_x(\omega)=\infty$ とする．列 $\omega x$ における最後の参照であるページ $x$ に対してページ落ちが発生するのは，$\mathbf{s}(\omega)$ の最初の $m$ 個の要素が記憶内容を構成するから，$D_x(\omega)>m$ であるとき，かつそのときに限るということが看取される．

さて，二つのスタック $\mathbf{s}(\omega)$ と $\mathbf{s}(\omega x)$ を考察し，$\mathbf{s}(\omega)$ におけるページの系列がどのように変えられることによって $\mathbf{s}(\omega x)$ が得られるのかを調べよう．最後に参照されたページは常に記憶領域内にあるはずであるから，$x\in S(1, \omega x)$ であることが分る．このことは，最後に参照されたページがスタックの先頭であることを意味する．すなわち，

$$D_x(\omega x)=1 \tag{6.4.2}$$

これは図6.4.1に示されている．

次に重要な特性は，参照されないページがスタックの上位に移動することは

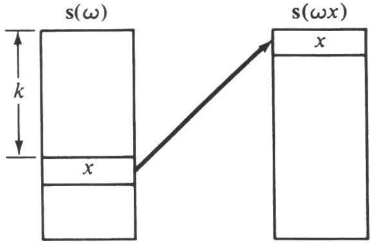



図 6.4-1　$D_x(\omega x)=1$ の図解

ないということである．すなわち，

(6.4.3)
$$D_y(\omega) \leq D_y(\omega x) \quad (y \neq x)$$

図 6.4-2 は，なぜこのことが成り立つかを示している．(a) と (b) で，ページ $y$ は $\mathbf{s}(\omega)$ の位置 $k+1$ にある，すなわち，$D_y(\omega)=k+1$ と仮定し，記憶領域が，丁度，$k$ ページから成る場合を想定する．今，$y$ が $\mathbf{s}(\omega x)$ の位置 $k$ あるいはもっと上位に移動するとしてみよう．(a) では，このことは，$x$ に対するページ落ちに際して，2 ページ ($x$ および $y$) が $k$ ページから成る記憶領域に入ったということを意味する．必要時ページングの下では，これは不可能である．次に，(b) では，$x$ に対するページ落ちが生じていないのに，ページ $y$ が $k$ ページから成る記憶領域に入ったことを示している．図 6.4-2 の (a)，(b) 両者共に起こり得ないことであるから，ページがスタックの上位に移動できるのは，それが参照されるとき，しかもそのときに限るという結論に到達する．

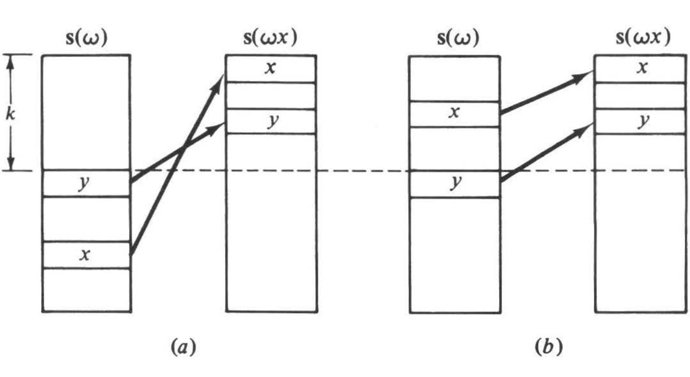



図 6.4-2　$D_y(\omega) \leq D_y(\omega x)$ の図解

スタック遷移を特徴付けるもう一つの特性は，参照されるページよりも下位にあるページはスタック内で同じ位置に留まるということである．すなわち，

$$(6.4.4) \qquad s_k(\omega x) = s_k(\omega) \quad (D_x(\omega) < k \text{ のとき})$$

**図 6.4-3** は，これがなぜ成り立たなければならないかを示している．条件(6.4.3)によって，ページ $y$ はより上位に移動することはできない．そこで，$y$ が $\mathbf{s}(\omega)$ において位置 $k$ にあり，記憶領域は $k$ ページから成る場合を考えよう．今，$y$ は $\mathbf{s}(\omega x)$ においてはもっと下位の位置を占めると仮定すると，参照ページ $x$ に対してページ落ちが発生しなかったにもかかわらず，$y$ は $k$ ページから成る記憶領域から排除されたものと結論せざるを得なくなる．このようなことがあってはならない．

**図 6.4-4** は，$\mathbf{s}(\omega)$ から $\mathbf{s}(\omega x)$ への遷移の効果を一つの図で示している．矢印はページの位置変化を示している．ページ $x$ は位置 $D_x(\omega)$ から位置1へ移動するので，$x$ より上位の位置にあったページは，条件(6.4.3)に合致するように，配置換えされなければならない．図で，$m$ ページから成る記憶領域の内容を $S$ と書くと，$S$ はスタックの上位の $m$ ページから成る．明らかに，$x$ の参照の結果として，$S$ 内のあるページ $u$ がそこから排除される．もしページ $y$ が初めにスタックの位置 $m+1$ にあったとしたならば，$x$ への参照の結果，同時に，$S+y$ 内のあるページ $v$ がそこから排除される．$v$ は $u$ か $y$ のどちらかであるから，$R(S+y, q, x)$ は $R(S, q, x)$ か $y$ のどちらかであるという重要な結果を得る．

このように述べることは，実際，図 6.4-4 の示す特徴を捉えていることが容易に頷けるであろう．したがって，スタック・アルゴリズムの次の特徴付けが得られる．

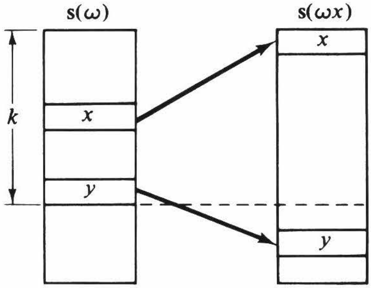



**図 6.4-3**　$s_k(\omega x) = s_k(\omega)$ の図解

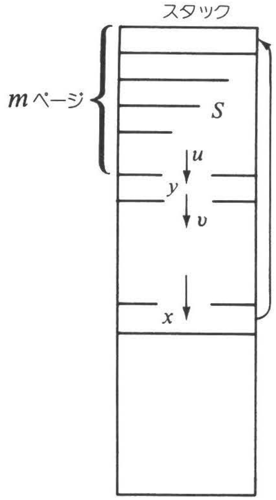



**図 6.4-4** スタック内のページの動き

**命題** 6.1

アルゴリズム $A$ は，次の条件が成り立つとき，かつそのときに限り，スタック・アルゴリズムである．

$R(S+y, q, x) \neq 0$，すなわち，$x \notin S+y$ であるときはいつでも次式が成り立つ．

(6.4.5) $$R(S+y, q, x) = R(S, q, x) \text{ または } y.$$

## 6.4.2 優先権アルゴリズム

6.2 節の各例は，排除ページ選択の決定を行うため，ページの集合 $N$ の上のある種の優先権順序付けを使用するアルゴリズムであった．(FIEO を除いて) これらのアルゴリズムが全てスタック・アルゴリズムであるということは，6.3.2 節および 6.4.1 節で既に言及した通りである．このことを明確にするため，まず，いわゆる優先権アルゴリズムのクラスを定義し，次に，条件(6.4.5)によってそれらが全てスタック・アルゴリズムであることを証明しよう．

ページング・アルゴリズム $A$ は，各参照列 $\omega = r_1 \ldots r_T$ に下記のような線形順序付けの系列 $\mathbf{p}_1 \ldots \mathbf{p}_{T-1}$ を対応付ける写像が存在するとき，**優先権アルゴリズム**(priority algorithm)であるという．

1. $\mathbf{p}_t (1 \leq t \leq T-1)$ は，$r_1 \ldots r_t$ 内に現れる相異なるページの順序付けであ

る.

2. 全ての $m \geq 1$ について次のことが成り立つ. すなわち, $r_{t+1} \notin S(m, r_1 ... r_t)$ かつ $|S(m, r_1 ... r_t)| = m (1 \leq t \leq T-1)$ であるならば, $S(m, r_1 ... r_t)$ から $A$ が排除するページは, $S(m, r_1 ... r_t)$ に含まれる $\mathbf{p}_t$ の最小元である.

$\min_{\mathbf{p}_t}[S]$ で $S$ に含まれる $\mathbf{p}_t$ の最小元を表すことにする. 記号 $\max_{\mathbf{p}_t}[S]$ も使用するが, 意味は明らかであろう. $\mathbf{p}_t$ であるということが了解できるのが普通であるから, 多くの場合, これを省略して書く. さて, $m$ および $A$ が了解されているとして, これまで使用してきた記号 $R$ を用いると, 上述の優先権アルゴリズムの定義により, 排除ページは次式で定義される.

$$(6.4.6)\qquad \begin{aligned} R(S_t, q, r_{t+1}) &= \min_{\mathbf{p}_t}[S_t] && (|S_t| = m,\ r_{t+1} \notin S_t) \\ &= 0 && (|S_t| < m \text{ または } r_{t+1} \in S_t) \end{aligned}$$

優先順位リストは, 制御状態 $q$ の一部という形で表現され, かつ維持される. 下記に示されるように, 各状態 $q$ がそれに対応する優先順位リスト $\mathbf{p}_t$ によって完全に決定されるように定式化できる重要な優先権アルゴリズムが存在する.

**例**: LRU, $B_0$, LFU, および LIFO アルゴリズムはいずれも優先権アルゴリズムであって, その優先順位リストは, それぞれ, 後向き距離の小さい順, 前向き距離の小さい順(および辞書式の順), 使用頻度の大きい順(および後向き距離の小さい順), および主記憶への到来時刻の早い順にページを順序付けている.

LRU の優先順位リストは LRU スタックに等しいということ, また, LRU と LIFO アルゴリズムは, 状態の集合 $Q$ が有限であるように定式化することができるということに注意しよう. 特に LRU アルゴリズムに対して, $Q$ を, $N$ から取った $m$ またはそれより少数の要素の全ての順列の集合と定義することができる. 各状態は, LRU スタックの最上位から $m$ またはそれより少数個を取って得たページであると解釈することができる. (このようにして, 態勢表示は, 制御状態だけで完全に定義される.) 対応する遷移関数は容易に構成することができる. 各参照に対して, 新しい状態は次のようにして形成される. すなわち, 参照したばかりのページをまずスタック(前の状態)の先頭に置く. それと同時に, 参照ページよりも上位にあった全てのページをプッシュダウンする(図6.4-7参照). もしも, 前の状態が $m$ ページから成っていて, 参照ページがその中のどれでもなかったとすると, この状態の最下位のページ

は新しい状態の中にはもはや存在しないということは明らかである．なお，LRU および LIFO アルゴリズムの有限状態的定式化とその遷移関数の細部に関しては，演習問題として残すことにする．最後に，LFU アルゴリズムは，それが非先見性であるという意味で実現可能であるけれども，使用頻度によるランク付けを維持するためには状態の無限集合を仮定しなければならないということを注意しておく．

式(6.4.5)を適用することにより，式(6.4.6)から，優先権アルゴリズムはスタック・アルゴリズムであるということが直ちに示される．特に，$x = r_{t+1} \notin S + y$，かつ $|S + y| = m$ であるならば，

$$
(6.4.7) \qquad \begin{aligned} R(S+y, q, x) &= \min_{\mathbf{p}_t}[S+y] \\ &= \min_{\mathbf{p}_t}[\min_{\mathbf{p}_t}[S], y] \\ &= \min_{\mathbf{p}_t}[R(S, q, x), y] \end{aligned}
$$

これから，$R(S+y, q, x)$ は，$R(S, q, x)$ または $y$ のどちらかとなる．また，逆も成立する．すなわち，各スタック・アルゴリズムと各参照列に対して，式(6.4.7)が真となるような優先順位リストの系列を定義することが可能である．非形式的ではあるが，このことを確かめるには，$\mathbf{s}_1, \ldots, \mathbf{s}_t, \ldots$ は，スタック・アルゴリズム $A$ が列 $r_1, \ldots, r_t, \ldots$ に対して生成するスタックの系列であるとし，$\mathbf{p}_t = \mathbf{s}_{t+1} (t = 1, 2, \ldots)$ で優先順位リストの系列を定義すればよい．これにより，$A$ は優先権アルゴリズムであることが分る．このことは，もし，$x$ が $S(m, \omega)$ に対してページ落ちを惹き起こすならば，$S(m, \omega)$ から排除されるページは $s_{m+1}(\omega x)$ になるという事実から得られる．換言すれば，$\mathbf{s}(\omega x)$ は，ページ排除が生じざるを得ない記憶領域の大きさ $m < D_x(\omega)$ の各々に対して，$S(m, \omega)$ の要素の上にページ排除の順位付けを与えている．明らかに，$D_x(\omega)$ よりも大きい距離を持つ要素に対する順位付けは重要でない．これ以上の詳細な議論に関しては文献[2]を当たるとよい．また，優先権アルゴリズムとスタック・アルゴリズムの等価性に関する形式的取扱いについては文献[13]を参照するとよい．

FIFO がスタック・アルゴリズムでないということは既に見た通りである．それにもかかわらず，6.2 節では，これを優先順位規則を用いて表現することができた．これは決して矛盾ではない．一般に，FIFO が用いる優先順位リストは，記憶領域の大きさに独立に定まるものではない．（ここで，優先順位リストというものは参照列だけの関数でなければならないということを想起する

必要がある.)表6.4-1は，FIFOがスタック・アルゴリズムでないことを明示しているが，$\omega = 123412$に対する記憶状態を考えよう．それを，記憶領域内への格納順序を示すベクトルとして表現すると，下記のようになる.

$$S(3, \omega) = (2, 1, 4)$$
$$S(4, \omega) = (4, 3, 2, 1)$$

ベクトル(2, 1, 4)では，2は4より先行するが，一方，ベクトル(4, 3, 2, 1)では，2は4より後に続くということが分る．このように優先順位に関する矛盾が生じるため，記憶領域の全ての大きさに対して適用可能な唯一の優先順位リストというものは存在しない．それゆえ，FIFOはどんな優先権アルゴリズムとも等価ではない.

興味を惹く排除ページ選択規則のあるものは，ある適用範囲内でランダム選択を行う規則を包含している．たとえば，RANDOMページング・アルゴリズムは排除ページを$S$からランダムに選択する．すなわち，各ページは選び出される機会を平等に持つ．また，互角解決の仕組をLRUでなくランダム選択とするLFU規則を考えることもできる．与えられた参照列の上でのこのような規則の効果は決定性ではないので，それに対応するアルゴリズムは本章で展開される形式化によっては記述できない．しかし，シミュレーションを行う目的で，擬似乱数ジェネレータを用いる優先権アルゴリズムを，上述した二つのようなランダム選択アルゴリズムと等価な統計的性質を持つように定義することができる[2]．特に，RANDOMアルゴリズムをシミュレートしたいと仮定しよう．擬似乱数ジェネレータを使って，$N$の部分集合のランダムな順列(線形順序付け)を生成する手続きを構成する．与えられた初期状態に対して，$\mathbf{I}_i$は，この手続きによって生成される$i$番目の順序付けを表すものとする．そこで，$\mathbf{p}_i = \mathbf{I}_i$という規約を設けることにより，目的に適った優先権アルゴリズムが定義されることになる．原理的には，各$y \in S \subseteq N$は$\min_{\mathbf{I}_i}[S]$となる機会を平等に持っている．したがって，この優先権アルゴリズムはRANDOMアルゴリズムに特有の統計的性質を持っているといえる．(しかしながら，最後に述べたことは，常に，擬似乱数ジェネレータによる制約を受ける.)さて，次にスタック更新手続きを記述することにしよう．その後で，RANDOMアルゴリズムのもっと簡単な実現が与えられる.

優先順位リストとスタックは一般に一致する必要はないから，ページング・アルゴリズム・シミュレータは，スタック・ベクトルと優先順位リストの両方を暗黙の内にであろうとあるいは顕在的にであろうといずれにしても維持しな

ければならない．そこで，優先順位リストからスタック・ベクトルを更新する方法が必要となる．この方法を**スタック更新手続き**(stack updating procedure)と呼ぶことにする．これは，定義 $R(S, q, x)=\min[S]$ を式(6.4.5)に適用し，式(6.4.2)—式(6.4.4)の諸性質を用いることによって得られる．

**命題 6.2(スタック更新手続き)**

$\mathbf{s}(\omega)$ と$\mathbf{s}(\omega x)$ は二つの引き続くスタックであるとして，$D_x(\omega)=m$ を仮定する．

このとき，次式が成り立つ．

$$
(6.4.8) \qquad s_i(\omega x)=\begin{cases} x & (i=1) \\ \max[s_i(\omega), \min[S(i-1, \omega)]] & (1<i<m) \\ \min[S(m-1, \omega)] & (i=m) \\ s_i(\omega) & (i>m) \end{cases}
$$

ただし，$S(k, \omega)=\{s_1(\omega), \ldots, s_k(\omega)\}$

上式の第1行は式(6.4.2)から，また，第4行は式(6.4.4)から得られる．第2行は，図6.4-4を観察することによって，スタック位置 $i$ に残されるページは，スタックの上位 $i-1$ ページの集合から排除された $\min[S(i-1, \omega)]$ と $s_i(\omega)$ のうちで上位 $i$ ページの集合からは排除されない方のページ，すなわち，$\max[s_i(\omega), \min[S(i-1, \omega)]]$ であることが分る．第3行も図6.4-4から得られる．$x$ が抜けて空になったスタック位置は，スタックにおいて $x$ より上位にあった $m-1$ ページの集合から排除されたページで充塡されなければならないからである．

式(6.4.8)で示される結果はそれにぴったりの図的表現を持っている．図6.4-5に示すように**優先権演算子**(priority operator)を定義する．これを用いる

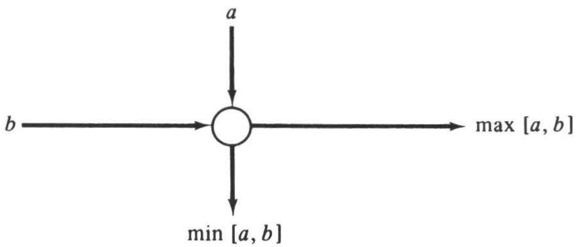



図6.4-5 優先権演算子

と，$D_x(\omega)=m$ に対する更新手続きは，**図6.4-6**のように表される．（一般に，$n_\omega$ が $\omega$ において参照される相異なるページの個数であるとすると，スタックは $n_\omega$ 個の項を含む．また，もし $D_x(\omega)=\infty$ であるならば，$\mathbf{s}(\omega x)$ は $n_\omega+1$ 個の項を含むことになり，$1\leq i\leq n_\omega+1$ に対するだけの更新ステップが遂行される.）

**例**：ベラディのアルゴリズム $B_0$ は，**表6.4-2**に与えられている参照列に対して同表に示されているような優先順位リストとスタックの系列を生成する．

**例**：前述したように，RANDOM ページング・アルゴリズムは，$N$ のある部分集合の順列の間からランダムに選ばれる優先順位リストに依存するものである．ところが，図6.4-6を考えてみると，このアルゴリズムのシミュレーションを極めて簡単に行うことができるということに気が付く．というのは，ランダムな意志決定を優先権演算子に持ち込むことだけしか必要としないからである．このアルゴリズムの一つの性質は，与えられたページが $m$ ページから成る記憶領域から選択される確率は $1/m$ であるということである．したがって，与えられたページ $x$ は確率 $1/m$ で $m$ 番目の優先権演算子に入る† ことになる．もしそれが同演算子を確率 $1/(m+1)$ で出る†† ことになるならば，その演算子における決定は次のように行われなければならない．すなわち，確

**表6.4-2**　$B_0$ スタック・アルゴリズム

| | $\omega$ | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 2 | 3 | 1 |
| **p** | 1 | 1 | 1 | 1 | 2 | 3 | 2 | 3 | 1 | 1 |
| | — | 2 | 2 | 2 | 3 | 2 | 3 | 1 | 2 | 2 |
| | — | — | 3 | 3 | 1 | 1 | 1 | 2 | 3 | 3 |
| | — | — | — | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| **s** | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 2 | 3 | 1 |
| | — | 1 | 1 | 1 | 4 | 1 | 2 | 3 | 2 | 2 |
| | — | — | 2 | 2 | 2 | 4 | 1 | 1 | 1 | 3 |
| | — | — | — | 3 | 3 | 3 | 4 | 4 | 4 | 4 |

†**訳注**　優先権演算子に**入る**(enter)ということは，図6.4-5で上から下へ向う矢線に沿って演算子を表す円に到達することに対応する．

††**訳注**　演算子から**出る**(leave)ということは，図6.4-5で上から下へ向う矢線に沿って演算子を表す円から出ることに対応する．

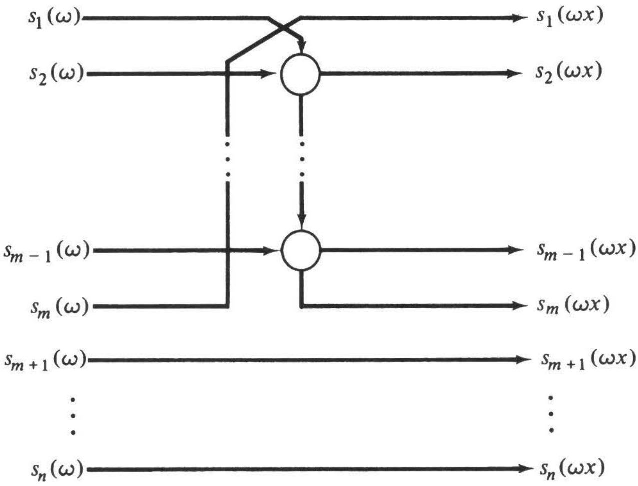



**図6.4-6** スタック更新手続き

率$1/(m+1)$ で $s_{m+1}$ を選択するか，または確率 $m/(m+1)$ でこの演算子に入って来るページ $x$ を選択するかのいずれかである.

先に，LRU アルゴリズムのスタックはその優先順位リストに等しいということを述べた．というのは，$S(m, \omega)$ は，$\omega$ における最後に参照された $m$ ページ，すなわち，$m$ 個の最高優先権ページを含むからである．命題6.2は，ページング・アルゴリズムのスタックと優先順位リストが各時点で相等しいのはそのアルゴリズムが LRU であるときに限るという逆命題の証明の基礎となる．帰納法の仮定として，$\mathbf{s}(\omega) = [s_1(\omega), \ldots, s_n(\omega)]$ は LRU スタックであること，したがって，$i<j$ はページ $s_i(\omega)$ が $\omega$ において $s_j(\omega)$ よりももっと最近に参照されたことを意味するということを仮定する．この仮定から，$\mathbf{s}(\omega x)$ も同様に順序付けられること，すなわち，$x = s_m(\omega)$ ならば，次のようになることを示せば十分である.

$$\text{(6.4.9)} \qquad \mathbf{s}(\omega x) = [x, s_1(\omega), \ldots, s_{m-1}(\omega), s_{m+1}(\omega), \ldots, s_n(\omega)]$$

$x = s_m(\omega)$ として命題6.2を適用すると，式(6.4.9)は容易に求められる．したがって，次の系が成り立つ.

**系 6.1**

スタック・アルゴリズムが LRU であるのは，そのスタックと優先順位リス

トが各ページ参照時に一致するときであり，かつそのときに限る．

**図6.4-7**は，対応するLRUスタック更新手続きを示す．後に示されるように，この結果は多重プログラミングに大変重要な係わりを持っている．

### 6.4.3　費用関数を決定するための手続き

スタック・アルゴリズムが重要であるのは，それが多くの実用的ページング・アルゴリズムのモデルになっているからだけでなく，いかなるスタック・アルゴリズム $A$ に対してもその費用関数 $C(A, m, \omega)$ を計算する効率的方法が存在するからでもある．記号を簡単にするため，$A$ が与えられていると仮定して，$m$ ページから成る記憶空間で $\omega$ を処理する際に $A$ によって発生する費用(すなわち，ページ落ち発生の回数)を $C(m, \omega)$ と書くことにする．

スタック・アルゴリズムの費用に関する第一に重要な事実は，$C(m, \omega)$ が **$m$ に関して非増加的である**ことである．すなわち，

(6.4.10)
$$C(m, \omega) \geq C(m+1, \omega)$$

この式は包含特性から直接に求められる．というのは，与えられた参照 $\omega$ に対して，$S(m+k, \omega)\,(k \geq 1)$ でページ落ちが発生することは，$S(m, \omega)\,(\subseteq S$

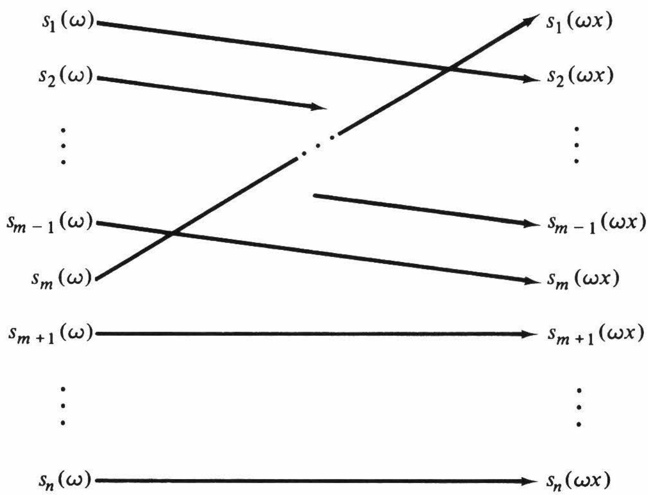



**図6.4-7**　LRUスタック更新

$(m+k, \omega)$)でもページ落ちが必ず発生することを意味するからである．FIFO アルゴリズムに対して6.3節で示した変則的振舞によって明示されるように，この性質は非スタック・アルゴリズムに対しては成立するとは限らない．

以前に導入したスタック距離の概念は，スタック更新手続きと相俟って，与えられた $\omega$ および全ての $m(1 \leq m \leq n)$ に対して $C(m, \omega)$ を計算する手続きの基礎を供与する．$A$ によって $\omega = r_1 r_2 \ldots r_t \ldots r_T$ が処理されて，その結果，スタック・ベクトルの系列 $\mathbf{s}_1 \mathbf{s}_2 \ldots \mathbf{s}_t \ldots \mathbf{s}_T$ が生成されると仮定しよう．$\mathbf{s}_{t-1}$ における $r_t$ の位置，すなわち，$\mathbf{s}_{t-1}$ における $r_t$ のスタック距離を $D_t$ と置くと†，$\omega$ に対応して，スタック距離の系列 $D_1 D_2 \ldots D_t \ldots D_T$ が定まることになる．時刻 $t$ にページ落ちが発生するのは，$D_t > m$ であるとき，しかもそのときに限るから，$C(m, \omega)$ を次式のように定義することができる．

(6.4.11) $$C(m, \omega) = |\{t | D_t > m, 1 \leq t \leq T\}|$$

$a_k$ は，スタック位置 $k$ にあったページがスタックの先頭に移される回数を表すと定義すると，

(6.4.12) $$a_k = |\{t | D_t = k, 1 \leq t \leq T\}|$$

と書くことができ，したがって，$C(m, \omega)$ を次式のように表現することができる．

(6.4.13) $$C(m, \omega) = a_\infty + \sum_{k=m+1}^{n} a_k$$

ただし，$a_\infty$ は，それまで参照されることのなかったページへの参照が行われた回数を表す．

上式は，スタックをある表現の仕方で維持し，かつ各スタック位置で行われた参照の回数をそれぞれ検出するだけで，$C(m, \omega)$ を $m$ の関数として決定することができるということを示唆している．手続きは，まず，ベクトル $\mathbf{a}$ が $\mathbf{a} = (a_1, \ldots, a_n, a_\infty) = (0, \ldots, 0, 0)$ と置かれることにより初期化される．参照 $r_t(1 \leq t \leq T)$ が処理される際には，そのスタック距離が $D_t$ であることが判明すると，$\mathbf{a}$ の要素 $a_{D_t}$ に1が加えられ，さらにスタックが更新される．これは $\omega$ の全部の参照が処理されてしまうまで繰返し行われる．手続きが終了すると，式(6.4.13)によりベクトル $\mathbf{a}$ から関数 $C(m, \omega)$ が算出されることになる．

†**訳注** ただし，$\mathbf{s}_0 = \phi$ とする

**例：表6.4-3** は，LRU アルゴリズムに対する二つの方法による $C(m, \omega)$ の計算を示す．$m=1, 2, 3, 4$ に対する記憶状態系列の数え上げともう一つは上記で概略を示した手続きを使用するものである．各 $m$ と $\omega$ における参照に対して，記憶状態は，**s** と同じにページを並べた(下方に向かって読んで行く)列ベクトルとして表現されている．ページ落ちの発生は記号 + を付して示してある．

## 6.5 拡 張 問 題 [4]

大きさ $m$ の記憶領域および所与の参照列に応じて，次のような取込み—排除の系列(fetch-replacement sequence)を生み出す必要時ページング・アルゴリズムを考えよう．

**表6.4-3**　LRU に対する費用計算

| | | $\omega$ | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 2 | 3 | 1 | $C$ |
| $m=1$ | — | 1+ | 2+ | 3+ | 4+ | 1+ | 2+ | 3+ | 2+ | 3+ | 1+ | 10 |
| $m=2$ | — | 1+ | 2+ | 3+ | 4+ | 1+ | 2+ | 3+ | 2 | 3 | 1+ | 8 |
| | — | — | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 2 | 3 | |
| $m=3$ | — | 1+ | 2+ | 3+ | 4+ | 1+ | 2+ | 3+ | 2 | 3 | 1 | |
| | — | — | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 2 | 3 | 7 |
| | — | — | — | 1 | 2 | 3 | 4 | 1 | 1 | 1 | 2 | |
| $m=4$ | — | 1+ | 2+ | 3+ | 4+ | 1 | 2 | 3 | 2 | 3 | 1 | |
| | — | — | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 2 | 3 | 4 |
| | — | — | — | 1 | 2 | 3 | 4 | 1 | 1 | 1 | 2 | |
| | — | — | — | — | 1 | 2 | 3 | 4 | 4 | 4 | 4 | |
| **p, s** | — | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 2 | 3 | 1 | |
| | — | — | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 2 | 3 | |
| | — | — | — | 1 | 2 | 3 | 4 | 1 | 1 | 1 | 2 | |
| | — | — | — | — | 1 | 2 | 3 | 4 | 4 | 4 | 4 | |
| $D$ | — | ∞ | ∞ | ∞ | ∞ | 4 | 4 | 4 | 2 | 2 | 3 | |
| **a** | | | | | | | | | | | | |
| $a_1$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 |
| $a_2$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 2 | 8 |
| $a_3$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 7 |
| $a_4$ | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 3 | 3 | 3 | 3 | 4 |
| $a_\infty$ | 0 | 1 | 2 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | — |

$$[(x, y)]_m = (x_1, y_1)(x_2, y_2)\ldots(x_i, y_i)\ldots \tag{6.5.1}$$

ここで，対$(x_i, y_i)(i>m)$は，$i$番目のページ落ち発生で$y_i$が記憶領域から**排出され**(pushed)(すなわち，排除され)，$x_i$が記憶領域に**充塡され**(pulled)(すなわち，格納され)たことを意味する．ただし，最初の$m$個の対は，記憶領域の初期状態は空であるので，充塡だけを表さなければならないから，$y_i=0$ $(1\leq i\leq m)$と定義する．このような系列$[(x, y)]_m$を**排出—充塡系列**(push-pull sequence)と呼ぶ．

排出—充塡系列によるページング・アルゴリズムの振舞の記述は，記憶状態系列$S_0S_1\ldots S_t\ldots$によるよりも弱い記述である．$t_1t_2\ldots t_i\ldots$は，ページ落ちが発生する(全ての)時刻の系列であるとしよう．すなわち，それは$S_t\neq S_{t-1}$が成り立つような時刻$t$(の全て)から成る系列である．もし次式

$$S_{t_i} = \begin{cases} S_{t_i-1} + x_i & (i\leq m) \\ S_{t_i-1} + x_i - y_i & (i>m) \end{cases} \tag{6.5.2}$$

が成立するように$(x_i, y_i)$を算出するならば，$\{S_t\}$から$[(x, y)]_m$が構成できることは明らかである．しかしまた，$[(x, y)]_m$が与えられたとき，構成することのできる$S_t(t\geq 1)$の諸状態は$\{S_{t_i}\}$に含まれるそれに他ならないということも明白である．**拡張問題**(extension problem)というのは非形式的には次のように述べられる．すなわち，ページング・アルゴリズムおよび**ページ落ち発生時にだけ記録される**状態に関連する情報(通常は$[(x, y)]_m$を含む)が与えられたとき，$k(k\geq 1)$ページ拡張された記憶領域に対する$[(x, y)]_{m+k}$が決定できるであろうか．これから示す通り，この問題はスタック・アルゴリズムに対しては可解である．というのは，包含特性は，$[(x, y)]_{m+k}$における充塡$(x_i)$の系列が$[(x, y)]_m$における充塡の系列の部分系列になることを意味するからである．

拡張問題は，ページ落ち発生時に記録された情報から，記憶領域を増大させたと仮定したときに，その結果得られるであろう性能向上の程度を突き止めたいという現実の問題によって動機付けられている．この問題は，ページ落ち発生時に無視できない程のシステム経費を加えなくても記録が取れるので，実用上の見地から興味を惹く．これに反し，優先順位リストまたはスタックの完全な記録を生成することは，プログラムの実行を参照毎に中断しなければならなくなるので，許されない程高く付く．

一般の(任意の)スタック・アルゴリズムに対して拡張問題を解くためには，

排除ページを決定するための優先順位リストを知ることが必要である．したがって，優先順位リストは，系列 $[(x, y)]_m$ と一緒に，ページ落ち発生時に記録される情報の中に含まれていなければならない．解は，スタックの距離 $m+1$ から $m+k$ までの部分の記録を維持することによって得られる．このスタック断片は式(6.4.8)で示される更新手続きを使って更新される．スタック(断片)を正しく更新することを可能にするためには，一見すると，優先順位リストを各参照後に欠かさずに知ることが必要ではないかと思いがちである．しかし，排出一充塡$(x_{i-1}, y_{i-1})$および$(x_i, y_i)$の間に行われる参照はスタック位置 $s_{m+1}, \cdots, s_m$ に影響をおよぼさないということは容易に看て取れる．中間に介在するこれらの参照はどれもページ落ちを惹起しないから，それらのスタック距離は $m$ を超えることがない．したがって，式(6.4.8)から，$m$ より大きい距離を持つスタック位置は不変である．

$[(x, y)]_m$ から $[(x, y)]_{m+k}$ を得るための手続きは，以下のような動作をする．各排出一充塡 $(x_i, y_i)$†に対して，次の二つの場合が考えられる．すなわち，1)$x_i$ のスタック距離が $m+k$ を超えるならば，$x_i$ はページ落ちを発生させ，$m+k$ ページの記憶領域からあるページ $y'_i$ を排出せしめるはずである．この場合は，$(x_i, y_i)$を次の段落で詳しく述べる$(x_i, y'_i)$で置き換える．2)$x_i$ のスタック距離が $m+1$ から $m+k$ までの範囲にあるならば，$m+k$ ページの記憶領域に対しては排出一充塡の対を生ぜしめない．この場合は，$[(x, y)]_m$ から$(x_i, y_i)$を削除する．以上のようにして $[(x, y)]_m$ の各項$(x_i, y_i)$を変換してしまうと，$[(x, y)]_m$ が $[(x, y)]_{m+k}$ に変換されたことになる．

第一の場合において$(x_i, y_i)$を$(x_i, y'_i)$に変換するために必要な計算を**図6.5-1(a)**に示す．この図は，$x_i$ のスタック距離が $m+k$ を超える場合に対応する図6.4-6の部分図である．スタック断片$(s_{m+1}, \ldots, s_{m+k})$の最新版を維持することが必須の条件となるから，そのために $[(x, y)]_m$ の各$(x_i, y_i)$の直前における優先順位リストを知る必要がこの計算にはある††．もしも，系列 $[(x, y)]_m$ だけしか与えられていないとするならば，拡張問題が可解であるアルゴリズムのクラスは著しく限定されることになる．優先順位リストを $[(x, y)]_m$ から導出し得るようなアルゴリズムの例を挙げると，LIFO と LRU がある．

LRU アルゴリズムの場合は，図6.4-7に示されているように，スタックと優先順位リストが一致しているから，拡張問題は特に簡単な解を持つ．$(s_{m+1},$

---

† $[(x, y)]_m$ は，与えられていると仮定するか，あるいはスタックの系列から導出されると仮定していることを注意しておく．

†† 実際に知る必要があるのは，ページ $y, s_{m+1}, \ldots, s_{m+k}$ の相対的優先順位だけである．

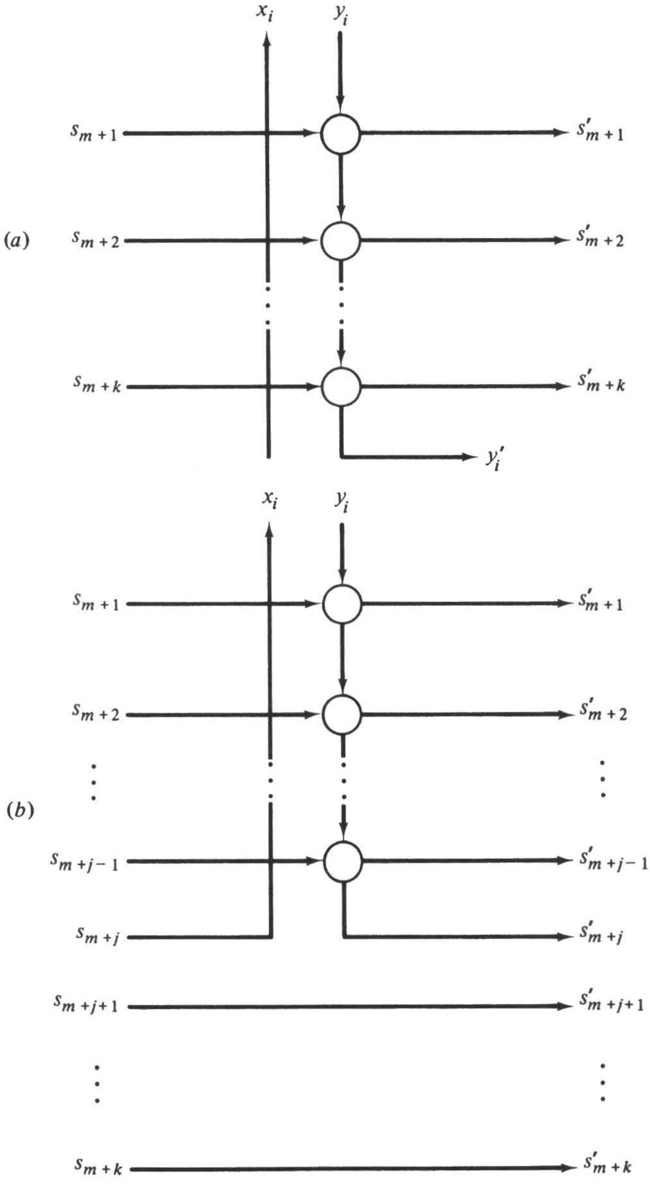



**図 6.5-1　拡張問題の計算**

…, $s_{m+k}$)は現在のスタック断片であるとし，また$(x_i, y_i)$は考察中の $[(x, y)]_m$ の排出一充塡対であるとしよう．LRU に対する拡張問題では，1ステップの計算は次の通りに行われる．

$$
\begin{aligned}
&j \leftarrow 1;\ done \leftarrow 0\\
&\textbf{while}(j \leq k)\textbf{and}(done = 0)\textbf{do}\\
&\quad \textbf{begin if } x_i = s_{m+j},\ \textbf{then } done \leftarrow 1;\\
&\qquad\quad \textbf{exchange}(s_{m+j},\ y_i)\ ;\ j \leftarrow j+1 \textbf{ end}\\
&\textbf{if } done = 0,\ \textbf{then output}(x_i,\ y_i)
\end{aligned}
$$

この手続きの $j$ 番目の繰返しの後に，$y_i$ が $m+j$ ページの記憶領域から排除されるページとなる．ある $j$ に対して $x_i = s_{m+j}$ が成立するならば，繰返し計算は終ってしまい，全ての $j$ に対して $x_i \neq s_{m+j}$ であるときにだけ，対$(x_i, y_i)$が出力される．

ここで提示した着想を適当に形式化すれば，与えられたページング・アルゴリズムが包含特性を持つとき，しかもそのときに限り拡張問題は可解であるということが困難なく示せる．この結果および拡張問題の様々な工学的な面に関する議論が文献[4]に見られる．記憶領域の大きさを増加させることに伴って生じる性能の向上を予測することが可能であるという実用的な重みが，ページング・システムの設計においてスタック・アルゴリズムを採用する理由を強化しているということを最後に注意しておく．

## 6.6　独立参照モデル

本節では，ページング・アルゴリズムの性能をページ落ち率の期待値を用いて決定する問題を考えよう．この目的で，**独立参照モデル**(independent reference model)として知られている単純な確率モデルを導入する．このモデルはプログラムの振舞の現実的表現にはなっていないと信じられる大きな理由がある．しかしながら，このモデルは次の二つの理由で重要である．第一に，このモデルは，その枠内で一つの最適アルゴリズムが明確に示される確率構造になっている(そのような構造がもう一つ6.7節で論じられることになっている)からであるし，第二に，このモデルでは，(ページング・アルゴリズムの)古典的な諸方法を用いたときのプログラムの振舞に対する解析(およびその複雑さ)が明瞭に示されるからである．

独立参照の仮定の下では，参照列 $r_1 r_2 \cdots r_t \cdots$ は，全ての $t \geq 1$ に対して $\Pr[r_t = i] = \beta_i$ となるような共通の定常分布 $\{\beta_1, \cdots, \beta_n\}$ に従う確率変数の系

列である．確率変数 $d_t(x)$ は $r_t$ 直後の $x$ の前向き距離を表すとすると，ここでの仮定により，$d_t(x)$ は，平均値 $\overline{d(x)} = 1/\beta_x$ を持つ定常的幾何分布

$$\Pr[d_t(x) = k] = \beta_x(1-\beta_x)^{k-1} \quad (k = 1, 2, \cdots) \tag{6.6.1}$$

に従うことになる．

さて，排除するべきページとして，記憶領域内に存在しているページの中で最大の前向き距離を持つ(すなわち，$\beta_x$ が最小である)ものを選択するアルゴリズムを $A_0$ と置く．$A_0$ はスタック・アルゴリズムであるということは容易に分る．ページは $\beta_1 \geq \beta_2 \geq \cdots \geq \beta_n$ が成り立つように番号付けられていると仮定するならば，次式のように $A_0$ のページ排除規則を簡単に表現することができる．

$$R(S, q, x) = S \text{ 内の最大整数} \tag{6.6.2}$$

$A_0$ は，次の意味で LIFO アルゴリズムに類似していることを注意しておく．$A_0$ によれば，長時間の後には，ページ $1, 2, \cdots, m-1$ は記憶領域内に存在することになると期待できる．確率 $\beta_1, \cdots, \beta_{m-1}$ は $j \geq m$ に対する確率 $\beta_j$ よりも大であるから，それらは，一度記憶領域内に入ると，もはや決して排除されることがない．一方，もし参照列の最初の相異なる $m-1$ 個の参照がページ $1, 2, \cdots, m-1$ に対するものであるならば，LIFO アルゴリズムを用いても，ある一定時間が経過した後ではページ $1, 2, \cdots, m-1$ が記憶領域内に保持され，それらは決して排除されることがない．

**定理 6.3**

独立参照の仮定の下で，$A_0$ は最適である．すなわち，全ての $m \geq 1$ に対して，$A_0$ は $C(A, m)$ を最小にする［式(6.3.4)を参照せよ］．

これを示すため，$S_t = S$ が与えられたとき，$r_{t+1} \cdots r_{t+k}$ を処理するのに要する費用の期待値の必要時ページングによって達成可能な最小値を $C_k(S)$ で表す．ただし，$C_0(S) = 0$ を仮定する．$x = r_{t+1}$ と置くと，次の最適性原理が得られる．

$$C_k(S) = \sum_{x \in S} \beta_x C_{k-1}(S) + \sum_{x \in N-S} \beta_x \{1 + \min_{y \in S}[C_{k-1}(S + x - y)]\} \tag{6.6.3}$$

$A_0$ は式(6.6.3)によって特徴付けられるということを示すならば，この定理は証明される．この証明は定理 6.2 のそれに酷似している(文献[1]を見よ)ので，ここでは省いて，演習問題として残すことにする．

**定理 6.4**

独立参照モデルでは，初期状態 $S_0$ を仮定したとき，極限ページ落ち率(参照当りの費用)が次式のように定義される．

$$F_{S_0} = \lim_{k\to\infty} \frac{C_k(S_0)}{k} \tag{6.6.4}$$

$A_0$ の下では，$F_{S_0}$ は $S_0$ に独立となり，次の式で与えられる．

$$F(A_0) = B - \sum_{i=m}^{n} \frac{{\beta_i}^2}{B} \tag{6.6.5}$$

ただし，記憶領域の大きさを $m$, $N = \{1, 2, \cdots, n\}$，$\beta_1 \geq \beta_2 \geq \cdots \geq \beta_n$，かつ

$$B = \sum_{i=m}^{n} \beta_i$$

とする．

**証明**　$A_0$ によって生成される記憶状態の系列 $S_1, \cdots, S_t, \cdots$ は確率変数の系列である．実際，$\{S_t\}$ は $S_0$ を初期状態として持つマルコフ連鎖になることが容易に分る．$t \geq 0$ および $x \in N$ に対して，$b_x(t) = \Pr[x \notin S_t]$ と定義すると，与えられた初期状態に対して費用の期待値を次のように表すことができる．

$$C_k(S_0) = \sum_{t=1}^{k} \Pr[r_t \notin S_{t-1}] = \sum_{t=1}^{k} \sum_{x=1}^{n} \beta_x b_x(t-1) \tag{6.6.6}$$

ここで，このマルコフ連鎖 $\{S_t\}$ では，$L_m = \{1, \cdots, m-1\}$ を含まない状態は非再帰的であり，その他の状態は全て再帰的であることを注意しておく．任意の初期状態 $S_0$ を仮定したとき，$S_t \supseteq L_m$ となる最も早い時刻を $t_0$ と置くと，ページ $x$ に対する参照の平均時間間隔は $1/\beta_x$ であるから，$t_0$ の平均値は有界であって，次式のように押えられる[†]．

$$E[t_0] \leq \sum_{x \in N} \frac{1}{\beta_x}$$

---

**[†]訳注**　任意の初期状態 $S_0$ を仮定したとき，$S_t \supseteq L_m$ となる最も早い時刻を確率変数 $T(S_0)$ で表す．$L_m(S_0) = L_m - S_0 = \{x_1, \cdots, x_l\}$ と置く．ページ $x_i \in L_m(S_0)$ を時刻 $t_0$ に参照して初めて記憶状態は $L_m$ を含むことになると仮定する．この仮定の下では，時刻 $t_0 - 1$ までにページ $x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_l$ は少くとも1回は参照されなければならない．ページ $j\,(j \neq x_i)$ が時刻 $t_0 - 1$ までに参照される回数を $k_j$ と置くと，この確率 $P_{t_0}(x_i, S_0)$ は↗

ところで，$r_1, \cdots, r_{t_0}$ におけるページ落ちの発生回数は $t_0$ 以下であるから，その平均値の上界は上式で与えられる．したがって，ある $S'_0 \supseteq L_m$ に対して

$$\lim_{k\to\infty} \frac{C_k(S_0)}{k} \leq \lim_{k\to\infty} \left[ \frac{E[t_0]}{k} + \frac{C_k(S'_0)}{k} \right]$$

となるので，任意の初期状態 $S_0$ に対してある状態 $S'_0 \supseteq L_m$ が存在し，

$$F_{S_0} = \lim_{k\to\infty} \frac{C_k(S'_0)}{k} \tag{6.6.7}$$

が成り立つということがいえる．そこで，$S_0 = L_m + z (z \in N - L_m)$を仮定し，この条件の下で $b_x(t)$ の計算を行うことにする．$b_x(t)$ を算定するため，まず，$b_x(t)$ を次のように表現する．

$$b_x(t) = \begin{cases} \Pr[x \notin S_{t-1} \text{ かつ } r_t \neq x] + \Pr[x \in S_{t-1} \text{ かつ } r_t \notin L_m + x] \\ \qquad\qquad\qquad\qquad\qquad (\text{ただし，} t > 0, m \leq x \leq n) \\ 0 \qquad\qquad\qquad\qquad\quad (\text{ただし，} t \geq 0, 1 \leq x \leq m-1) \end{cases}$$

引き続く参照が互いに独立であることを用いると，次式が得られる．

$$b_x(t) = b_x(t-1)(1-\beta_x) + [1 - b_x(t-1)](B - \beta_x)$$
$$(t > 0, m \leq x \leq n)$$

すなわち，

$$b_x(t) = b_x(t-1)(1-B) + (B - \beta_x) \quad (t > 0, m \leq x \leq n) \tag{6.6.8}$$

---

↘ $$P_{t_0}(x_i, S_0) = \sum_{\substack{k_1+\cdots k_{x_i-1}+k_{x_i+1}+\cdots+k_n=t_0-1 \\ k_{x_j}\geq 1(j=1,\cdots,i-1,i+1,,\cdots,l)}} \frac{(t_0-1)!}{k_1!\cdots k_{x_i-1}!k_{x_i+1}!\cdots k_n!} \beta_1^{k_1}\cdots\beta_{x_i-1}^{k_{x_i-1}}\beta_{x_i+1}^{k_{x_i+1}}\cdots\beta_n^{k_n}\beta_{x_i}$$
$$\leq \sum_{k_1+\cdots+k_{x_i-1}+k_{x_i+1}+\cdots+k_n=t_0-1} \frac{(t_0-1)!}{k_1!\cdots k_{x_i-1}!k_{x_i+1}!\cdots k_n!} \beta_1^{k_1}\cdots\beta_{x_i-1}^{k_{x_i-1}}\beta_{x_i+1}^{k_{x_i+1}}\cdots\beta_n^{k_n}\beta_{x_i}$$
$$= (\beta_1 + \cdots + \beta_{x_i-1} + \beta_{x_i+1}\cdots + \beta_n)^{t_0-1}\beta_{x_i}$$
$$= (1-\beta_{x_i})^{t_0-1}\beta_{x_i}$$

上記で仮定した事象は，異なる $x_i$ に対しては互いに排反する事象となるから，初期状態 $S_0$ を仮定して，時刻 $t_0$ に初めて記憶状態が $L_m$ を含むことになる確率は

$$P_{t_0}(S_0) = \sum_{x_i \in L_m(S_0)} P_{t_0}(x_i, S_0) \leq \sum_{x_i \in L_m(S_0)} (1-\beta_{x_i})^{t_0-1}\beta_{x_i}$$
$$\leq \sum_{x \in N} (1-\beta_x)^{t_0-1}\beta_x$$

となり，$T(S_0)$ の平均値は

$$E[T(S_0)] = \sum_{t_0=0}^{\infty} t_0 P_{t_0}(S_0) \leq \sum_{x \in N} \frac{1}{\beta_x}$$

である．この結果から分るように，$T(S_0)$ の平均値は初期状態 $S_0$ に無関係に上界 $\sum_{x \in N}(1/\beta_x)$を持つ．

ただし，

$$B=\sum_{i=m}^{n}\beta_i$$

式(6.6.8)を系統的に解いて行くと，次式が得られる．

$$b_x(t)=\left[b_x(0)-\left(1-\frac{\beta_x}{B}\right)\right](1-B)^t+\left(1-\frac{\beta_x}{B}\right)\quad(t>0,\ m\leq x\leq n)$$

ただし，

$$b_x(0)=\begin{cases}1 & (x\neq z)\\ 0 & (x=z)\end{cases}$$

これを式(6.6.6)に代入すると，$t\geq 0$ かつ $1\leq x\leq m$ に対しては $b_x(t)=0$ であるので，

$$(6.6.9)\quad C_k(S_0)=\sum_{x=m}^{n}\beta_x\left\{\left[b_x(0)-\left(1-\frac{\beta_x}{B}\right)\right]\sum_{t=0}^{k-1}(1-B)^t+k\left(1-\frac{\beta_x}{B}\right)\right\}$$

が得られる．総和 $\sum_{t=0}^{k-1}(1-B)^t$ は上界 $1/B$ によって押えられているから，

$$(6.6.10)\qquad \lim_{k\to\infty}\frac{C_k(S_0)}{k}=\sum_{x=m}^{n}\beta_x\left(1-\frac{\beta_x}{B}\right)$$

を得る．式(6.6.5)はこの式からただちに得られる．

続いて，独立参照の仮定の下で，LRU ページング・アルゴリズムの解析を有限マルコフ連鎖の理論を用いて行い，さらに，この解析を拡張して，どのようにFIFO アルゴリズムに適用するかを示すことにしよう．

考察の対象となるマルコフ連鎖を $\{\mathbf{s}_i\}$ $(i=0, 1, 2, \cdots)$ と置く．このマルコフ連鎖は，記憶領域の大きさが $m$ であり，また集合 $N$ が $n$ ページから成るという仮定の下で，参照列 $r_1, \cdots, r_i, \cdots$ に対して LRU アルゴリズムが生成するスタックの系列に対応する．正確に述べると，このマルコフ連鎖の状態は LRU スタックの先頭から上位 $m$ 個の要素である．したがって，$N$ から取った $m$ 個の要素の順列の全てから成る集合が状態の集合である．$\{\mathbf{s}_i\}$ のマルコフ性は容易に確かめることができる．$N$ から取った $m$ ページから成る任意の順列を初期状態 $\mathbf{s}_0$ として選ぼう．ここでの目的は平衡状態におけるページ落ち率を調べることにあるから，この仮定は便宜的なものである．系列 $\{\mathbf{s}_i\}$ に関する諸仮定によって，この初期条件は結果に何らの影響も及ぼすことはない．

マルコフ連鎖 $\{\mathbf{s}_i\}$ は，1段遷移確率

$$p(\mathbf{s}, \mathbf{s}') = \Pr[\mathbf{s}_i = \mathbf{s}' | \mathbf{s}_{i-1} = \mathbf{s}] \quad (i \geq 1)$$

が明らかに時間パラメータ $i$ に独立に定まるという意味で一様(時間的一様)である．遷移確率は，LRU スタックの可能な遷移(図 6.4-7 参照)から容易に求めることができる．特に，$\mathbf{s} = [j_1, \cdots, j_m]$，$k \neq j_i (1 \leq i \leq m)$，かつ $\mathbf{s}' = [k, j_1, \cdots, j_{m-1}]$ (ページ $k$ への参照の結果得られるスタック)ならば，$p(\mathbf{s}, \mathbf{s}') = \beta_k$ である．また，$k = j_i$，かつ $\mathbf{s}' = [k, j_1, \cdots, j_{i-1}, j_{i+1}, \cdots, j_m]$ ならば，このときも $p(\mathbf{s}, \mathbf{s}') = \beta_k$ である．これ以外の遷移のあり方はないから，前出の関係を満たさない全ての $\mathbf{s}$ と $\mathbf{s}'$ に対しては $p(\mathbf{s}, \mathbf{s}') = 0$ となる．

$\{\mathbf{s}_i\}$ が**既約**(irreducible)であること，すなわち，各 $\mathbf{s}$ および各 $\mathbf{s}'$ に対して，状態 $\mathbf{s}$ から状態 $\mathbf{s}'$ へ $k$ 回の遷移で移行する確率 $p^{(k)}(\mathbf{s}, \mathbf{s}')$ が零でないような正の数 $k$ が存在することを示すのは困難なことではない．スタックの任意の対 $\mathbf{s} = [j_1, \cdots, j_m]$ および $\mathbf{s}' = [j'_1, \cdots, j'_m]$ に対して，特に，参照の系列 $j'_m, \cdots, j'_1$ は $\mathbf{s}$ から $\mathbf{s}'$ へ $m$ 段の遷移で移行せしめることは明白である．このことは，全ての $\mathbf{s}$ と $\mathbf{s}'$ に対して $p^{(m)}(\mathbf{s}, \mathbf{s}') = \prod_{i=1}^{m} \beta_{j'_i} > 0$ を意味する．したがって，連鎖 $\{\mathbf{s}_i\}$ は既約である．この結果，平衡方程式を一意に満たす定常確率分布が存在することになる．平衡状態確率ベクトルを $\boldsymbol{\pi}$，遷移確率 $p(\mathbf{s}, \mathbf{s}')$から成る行列を P と置くと，平衡方程式は次の通りである．

$$\boldsymbol{\pi} = \boldsymbol{\pi}\mathsf{P} \tag{6.6.11}$$

状態 $\mathbf{s}$ の平衡状態確率を $\pi_{\mathbf{s}} (\mathbf{s} = [j_1, \cdots, j_m])$ で表すと，正規化条件は次のように書くことができる．

$$\sum_{\mathbf{s} \in Q} \pi_{\mathbf{s}} = 1 \tag{6.6.12}$$

現在の記憶状態が $\mathbf{s}$ であるという条件の下で，ページ落ちが発生する平衡状態確率は $p_f(\mathbf{s})$ であるとする．$\pi_{\mathbf{s}}$ は定常状態における状態 $\mathbf{s}$ の生起頻度であると解釈すると，状態 $\mathbf{s}$ からのページ落ち発生率は $p_f(\mathbf{s})\pi_{\mathbf{s}}$ であるので，統計的平衡状態における全ページ落ち率は

$$F(\mathrm{LRU}) = \sum_{\mathbf{s} \in Q} p_f(\mathbf{s}) \pi_{\mathbf{s}} \tag{6.6.13}$$

となる．$F(\mathrm{LRU})$ に関し，次の結果が得られている．

**定理** 6.5

独立参照モデルでは，

$$(6.6.14)\qquad F(\mathrm{LRU})=\sum_{\mathbf{s}\in Q}D_1{}^2(\mathbf{s})\prod_{i=1}^{m}\frac{\beta_{j_i}}{D_i(\mathbf{s})}$$

が成り立つ．ただし，$\mathbf{s}=[j_1,\cdots,j_m]$，また，

$$(6.6.15)\qquad D_i(\mathbf{s})=1-\sum_{k=1}^{m-i+1}\beta_{j_k}$$

**証明**　$D_i(\mathbf{s})$ の定義により，$p_f(\mathbf{s})=D_1(\mathbf{s})$ であることが直ちにいえる．したがって，残されていることは，式(6.6.11)を満たす状態確率の表現式を求めることだけである．LRU システムに対して，式(6.6.12)を満足する式(6.6.11)の解は次の通り与えられる．

$$(6.6.16)\qquad \pi_{\mathbf{s}}=\frac{\prod_{i=1}^{m}\beta_{j_i}}{\prod_{i=2}^{m}D_i(\mathbf{s})}$$

ただし，$D_i(\mathbf{s})$ は式(6.6.15)で与えられる．

式(6.6.16)を証明するには，まず，それが平衡方程式(6.6.11)を満足することを示さなければならない．連立方程式(6.6.11)から状態 $\mathbf{s}=[j_1,j_2,\cdots,j_m]$ に対する式を抜き出せば，この式は，LRU 規則によって次の通り与えられる†．

$$\pi_{\mathbf{s}}=\beta_{j_1}\Big[\sum_{\substack{j\neq j_i\\(1\le i\le m)}}\pi_{[j_2,\cdots,j_m,j]}+\sum_{1\le k\le m}\pi_{[j_2,\cdots,j_k,j_1,j_{k+1},\cdots,j_m]}\Big]$$

式(6.6.16)を右辺に代入すると，

$$(6.6.17)\qquad [\text{右辺}]=\beta_{j_1}\left[\sum_{\substack{j\neq j_i\\(1\le i\le m)}}\frac{\beta_j\prod_{i=2}^{m}\beta_{j_i}}{\prod_{i=2}^{m}D_i(\mathbf{q}_j)}+\sum_{1\le k\le m}\frac{\prod_{i=1}^{m}\beta_{j_i}}{\prod_{i=2}^{m}D_i(\mathbf{r}_k)}\right]$$

ここで，$\mathbf{q}_j=[j_2,j_3,\cdots,j_m,j]$，$\mathbf{r}_k=[j_2,\cdots,j_k,j_1,j_{k+1},\cdots,j_m]$．ところで，

†**訳注**　下記で，$[j_2,\cdots,j_k,j_1,j_{k+1},\cdots,j_m]$ は，$k=1$ のとき $[j_1,j_2,\cdots,j_m]$ を表し，$k=m$ のとき $[j_2,\cdots,j_m,j_1]$ を表す．

(6.6.18) $$D_i(\mathbf{q}_j) = 1 - \sum_{k=2}^{m-k+2} \beta_{j_k} = D_{i-1}(\mathbf{s}) + \beta_{j_1} \quad (i > 1)$$

同様に,

(6.6.19) $$D_i(\mathbf{r}_k) = \begin{cases} D_i(\mathbf{s}) & (i \leq m-k+1) \\ D_{i-1}(\mathbf{s}) + \beta_{j_1} & (i > m-k+1) \end{cases}$$

であるから，これらを式(6.6.17)に代入して得られる式を整理すると†,

(6.6.20) $$[\text{右辺}] = \prod_{i=1}^{m} \beta_{j_i} \left[ \sum_{\substack{j \neq j_i \\ (1 \leq i \leq m)}} \frac{\beta_j}{\prod_{i=1}^{m} [D_i(\mathbf{s}) + \beta_{j_1}]} + \sum_{1 \leq k \leq m} \frac{\beta_{j_1}}{\prod_{i=2}^{k} D_i(\mathbf{s}) \prod_{i=k}^{m} [D_i(\mathbf{s}) + \beta_{j_1}]} \right]$$

通分し，$\sum_{\substack{j \neq j_i \\ (1 \leq i \leq m)}} \beta_j = D_1(\mathbf{s})$ を使うと，式(6.6.20)を次のように変形することができる.

(6.6.21)
$$[\text{右辺}] = \frac{\prod_{i=1}^{m} \beta_{j_i}}{\prod_{i=2}^{m} D_i(\mathbf{s})} \left[ \frac{\prod_{i=1}^{m} D_i(\mathbf{s}) + \beta_{j_1} \sum_{1 \leq k \leq m} \prod_{i=k+1}^{m} D_i(\mathbf{s}) \prod_{i=1}^{k-1} [D_i(\mathbf{s}) + \beta_{j_1}]}{\prod_{i=1}^{m} [D_i(\mathbf{s}) + \beta_{j_1}]} \right]$$

---

†**訳注** まず，$D_m(\mathbf{s}) + \beta_{j_1} = (1 - \beta_{j_1}) + \beta_{j_1} = 1$ であることに注意すると,

$$\prod_{i=2}^{m} D_i(\mathbf{q}_j) = \prod_{i=1}^{m-1} [D_i(\mathbf{s}) + \beta_{j_1}] = \prod_{i=1}^{m} [D_i(\mathbf{s}) + \beta_{j_1}]$$

同様に,

$$\sum_{1 \leq k \leq m} \frac{1}{\prod_{i=2}^{m} D_i(\mathbf{r}_k)} = \sum_{1 \leq k \leq m} \frac{1}{\prod_{i=2}^{m-k+1} D_i(\mathbf{s}) \cdot \prod_{i=m-k+2}^{m} [D_{i-1}(\mathbf{s}) + \beta_{j_1}]}$$
$$= \sum_{1 \leq k' \leq m} \frac{1}{\prod_{i=2}^{k'} D_i(\mathbf{s}) \cdot \prod_{i=k'+1}^{m} [D_{i-1}(\mathbf{s}) + \beta_{j_1}]}$$
$$= \sum_{1 \leq k \leq m} \frac{1}{\prod_{i=2}^{k} D_i(\mathbf{s}) \cdot \prod_{i=k}^{m} [D_i(\mathbf{s}) + \beta_{j_1}]}$$

等式(証明は容易である[†])

$$\prod_{i=1}^{m} x_i = \prod_{i=1}^{m}(x_i+\lambda) - \lambda \sum_{1\le k\le m} \prod_{i=1}^{k-1}(x_i+\lambda)\prod_{i=k+1}^{m} x_i$$

を用いると，式(6.6.21)右辺の外側の大括弧で括られた因数は1になるから，式(6.6.16)により $\pi_s$ に等しいということがいえる．したがって，式(6.6.16)は平衡方程式(6.6.11)の解であるという結論が得られる．

式(6.6.16)が式(6.6.12)の正規化条件を満足することは，$m$ に関する帰納法によって証明することができる．章末の演習問題(問題6.3)として残すことにするので，その詳細証明は読者にまかせる[††]．最後に，$p_f(\mathbf{s}) = D_1(\mathbf{s})$ と式(6.6.16)の $\pi_s$ を式(6.6.13)に代入することにより，求める結果が得られる．■

続いて，前述の解析をFIFOアルゴリズムに適用しよう．FIFOアルゴリ

---

**†訳注** 帰納法によって証明する．$m=1$ のとき，

$$[右辺] = (x_1+\lambda) - \lambda\prod_{i=1}^{0}(x_i+\lambda)\prod_{i=2}^{1} x_i = (x_1+\lambda)-\lambda = x_1 = [左辺]$$

$m<n$ のとき成立すると仮定して，$m=n$ のとき成り立つことを示す．

$$\begin{aligned}\prod_{i=1}^{n} x_i &= (x_n+\lambda-\lambda)\left[\prod_{i=1}^{n-1}(x_i+\lambda) - \lambda\sum_{1\le k\le n-1}\prod_{i=1}^{k-1}(x_i+\lambda)\prod_{i=k+1}^{n-1}x_i\right]\\ &= \prod_{i=1}^{n}(x_i+\lambda) - \lambda\prod_{i=1}^{n-1}(x_i+\lambda) - \lambda x_n\sum_{1\le k\le n-1}\prod_{i=1}^{k-1}(x_i+\lambda)\prod_{i=k+1}^{n-1}x_i\\ &= \prod_{i=1}^{n}(x_i+\lambda) - \lambda\prod_{i=1}^{n-1}(x_i+\lambda)\prod_{i=n+1}^{n}x_i - \lambda\sum_{1\le k\le n-1}\prod_{i=1}^{k-1}(x_i+\lambda)\prod_{i=k+1}^{n}x_i\\ &= \prod_{i=1}^{n}(x_i+\lambda) - \lambda\sum_{1\le k\le n}\prod_{i=1}^{k-1}(x_i+\lambda)\prod_{i=k+1}^{n}x_i\end{aligned}$$

**††訳注** しかしながら，全体の証明を完結させるため，この証明をも示しておく，$\mathbf{s}=[j_1,\cdots,j_m]$ に対して，$\mathbf{s}_{j_{m+1}} = [j_1,\cdots,j_m,j_{m+1}]$ (ただし，$j_{m+1}\notin\{j_1,\cdots,j_m\}$)と置くと，

$$D_i(\mathbf{s}_{j_{m+1}}) = 1-\sum_{k=1}^{(m+1)-i+1}\beta_{j_k} = 1-\sum_{k=1}^{m-(i-1)+1}\beta_{j_k} = D_{i-1}(\mathbf{s})$$

であるから，

$$\pi_{\mathbf{s}j_{m+1}} = \frac{\prod_{i=1}^{m+1}\beta_{j_i}}{\prod_{i=2}^{m+1}D_i(\mathbf{s}_{j_{m+1}})} = \frac{\beta_{j_{m+1}}\prod_{i=1}^{m}\beta_{j_i}}{D_1(\mathbf{s})\prod_{i=2}^{m}D_i(\mathbf{s})} = \frac{\beta_{j_{m+1}}}{D_1(\mathbf{s})}\cdot\pi_{\mathbf{s}}$$

したがって，

$$\begin{aligned}\sum_{\mathbf{s}\in Q}\sum_{j_{m+1}\notin\{j_1,\cdots,j_m\}}\pi_{\mathbf{s}j_{m+1}} &= \sum_{\mathbf{s}\in Q}\frac{\sum_{j_{m+1}\notin\{j_1,\cdots j_m\}}\beta_{j_{m+1}}}{D_1(\mathbf{s})}\cdot\pi_{\mathbf{s}}\\ &= \sum_{\mathbf{s}\in Q}\frac{1-\sum_{i=1}^{m}\beta_{j_i}}{D_1(\mathbf{s})}\cdot\pi_{\mathbf{s}} = \sum_{\mathbf{s}\in Q}\pi_{\mathbf{s}} = 1\end{aligned}$$

($\pi_{[j]}=\beta_j$ であるから，$m=1$ のとき成り立つことは明白.)以上により，$\pi_{\mathbf{s}}$ は正規化条件を満足することが証明された．

ズムに対しては，マルコフ連鎖の状態はLRUスタックの先頭からの上位 $m$ 個の要素の代りに $m$ 個の要素から成るFIFOリスト(行列)に対応する．$j_1$ を行列の先頭に持つ(すなわち，$j_1$ が記憶領域内に最も長い時間滞留している)FIFO行列を状態 $\mathbf{s}=[j_1, \cdots, j_m]$ で表す．このように状態を定義すると，参照 $r_i$ によって生じる可能性のある遷移は次の通りである．

$$[j_1, \cdots, j_m] \longrightarrow [j_1, \cdots, j_m] \quad (r_i=j_l, 1 \leq l \leq m)$$
$$[j_1, \cdots, j_m] \longrightarrow [j_2, \cdots, j_m, k] \quad (r_i=k \neq j_l, 1 \leq l \leq m)$$

任意に与えられた状態から他の任意の状態へ移行する遷移の系列を示すことによって $\{\mathbf{s}_i\}$ が既約であることを証明するのは，ここでも困難なことではない(演習問題6.6でこれを証明せよ)．また，LRUアルゴリズムに対する場合と同じくここでも，$m$ 個のページの任意の順列を初期状態として仮定しよう．

$F$(FIFO) を求めるため，前と同様に，式(6.6.11)に相当する連立方程式を解き，解を式(6.6.13)に相当する式に代入するという方法を推し進めよう．

**定理 6.6**

独立参照モデルでは，

$$F(\text{FIFO})=G^{-1}\sum_{\mathbf{s}\in Q} D_1(\mathbf{s}) \prod_{i=1}^{m} \beta_{j_i} \tag{6.6.22}$$

が成り立つ．ただし，$\mathbf{s}=[j_1, \cdots, j_m]$，かつ，

$$G=\sum_{\mathbf{s}\in Q} \prod_{i=1}^{m} \beta_{j_i} \tag{6.6.23}$$

**証明** 前述の状態遷移に従って，$\mathbf{s}=[j_1, \cdots, j_m]$ に対する平衡方程式は次の通り与えられる．

$$\pi_{\mathbf{s}}=\pi_{\mathbf{s}}\sum_{i=1}^{m} \beta_{j_i}+\sum_{\substack{j\neq j_i \\ (1\leq i\leq m)}} \beta_{j_m}\pi_{[j, j_1, \cdots j_{m-1}]} \tag{6.6.24}$$

式(6.6.24)に仮定解

$$\pi_{\mathbf{s}}=G^{-1}\prod_{i=1}^{m} \beta_{j_i} \tag{6.6.25}$$

を代入する．ただし，$G$ は式(6.6.23)によって与えられる．代入の結果，

$$[\text{式(6.6.24)の右辺}]=\pi_{\mathbf{s}}[1-D_1(\mathbf{s})]+\sum_{\substack{j\neq j_i \\ (1\leq i\leq m)}} \beta_{j_m}\frac{\beta_j \prod_{i=1}^{m-1} \beta_{j_i}}{G}$$

$$=\pi_s[1-D_1(\mathbf{s})]+\pi_s\sum_{\substack{j\neq j_i\\(1\leq i\leq m)}}\beta_j=\pi_s=[\text{式(6.6.24)の左辺}]$$

となるから，式(6.6.25)は確かに方程式(6.6.24)の解である．さらに，式(6.6.25)が正規化条件(6.6.12)を満たすことは，式(6.6.25)の形から明白である．以上により，式(6.6.25)が FIFO アルゴリズムに対する平衡方程式(6.6.11)の解であるという結論が得られる．最後に，式(6.6.25)を

$$F(\text{FIFO})=\sum_{\mathbf{s}\in Q}D_1(\mathbf{s})\pi_s$$

に代入することにより，式(6.6.22)が得られる．

本節では，独立参照モデルを用いたときの結果を，$A_0$，LRU，および FIFO アルゴリズムのみに対して求めたが，King[3] は，上述の技法を適用できるページング・アルゴリズムのクラスの構造を形式化している．また，ページングの振舞の基本的なモデルを一般化して，$r_t$ に対するページ参照確率が $r_{t-1}$ の参照直後の状態に(しかもそれのみに)依存するという仮定を包含するようにすることも困難ではない．そうすることによって，解析の対象である状態系列のマルコフ性を失うことも特にない．独立参照モデルに対するページ落ち率の上(下)界に関する結果については，演習問題(6.11〜6.14)で扱うことにする．

## 6.7 LRU-スタック・モデル

本節では，**LRU-スタック・モデル**を研究しよう．このモデルは，実際の参照列の統計的特性に極めて近い特性を持つ参照列を生成する能力を持つということが知られており[14]，様々な研究者によってかなりに詳細にわたる研究がなされてきた．最初に Shemer 等[17, 18]によって提案されたことがその発端

6.4 節では，参照列 $r_1r_2\cdots r_t\cdots$ に LRU スタックの系列 $\mathbf{s}_0\mathbf{s}_1\cdots\mathbf{s}_t\cdots$ をどのようにして対応付けるかを示した．$\omega=r_1r_2\cdots r_t$ に対応する LRU スタックは，要素 $s_i(\omega)$ が最後から数えて $i$ 番目の最終参照時刻を持つページであるようなベクトル $\mathbf{s}(\omega)=[s_1(\omega),\cdots,s_n(\omega)]$ である．すなわち，$\mathbf{s}(\omega)$ は後向き距離の小さい順にページを順序付けている．$D_x(\omega)$ は，スタック $\mathbf{s}(\omega)$ におけるページ $x$ の位置，すなわち，$x$ のスタック距離を表すとすると，参照列に**距離列**(distance string)$D_1D_2\cdots D_t\cdots$ を対応付けることが可能であった．ここで，$D_t$ はスタック $\mathbf{s}_{t-1}$ における $r_t$ の距離である．表現を変えると，$D_t$ は，ページ $r_t$ への最後の参照以後に参照された相異なるページの数である．

参照列生成の LRU スタック・モデルでは，LRU スタック距離列を次のよ

うな定常確率塊関数 $\{a_i\}$ に従う独立な確率変数の系列と考える．すなわち，

$$\Pr[D_t = i] = a_i \quad (i = 1, 2, \cdots, n;\ t \geq 1) \tag{6.7.1}$$

その累積分布関数は次式で与えられる．

$$A_i = \sum_{j=1}^{i} a_j \tag{6.7.2}$$

もっと厳密に述べるならば次の通りである．便宜的に† $\mathbf{s}_0 = [1, 2, \cdots, n]$ と置くことができる初期スタック $\mathbf{s}_0$ が与えられていると仮定して，確率変数 $D_t$ の見本値 $i_t(t = 1, 2, \cdots)$ の系列 $\mathbf{i} = i_1 i_2 \cdots i_t \cdots$ が与えられたとき，これに対応する LRU スタックの系列を $\mathbf{s}_0\mathbf{s}_1\cdots\mathbf{s}_t\cdots$（ただし，$\mathbf{s}_t = [s_t(1), \cdots, s_t(n)]\ (t = 1, 2, \cdots)$）としよう．次式によって定義される列 $\omega = r_1 r_2 \cdots r_t \cdots$ を $\mathbf{i}$ によって生成される参照列という．

$$r_t = s_t(1) \quad (t \geq 1) \tag{6.7.3}$$

初期スタック $\mathbf{s}_0$ と確率塊関数 $\{a_i\}$ が上述のように与えられていると了解したとき，上記のようにして定義される列の族を **LRU 参照列**(LRU reference string)の族と呼ぶことにする．与えられた確率塊関数 $\{a_i\}$ によって定義された LRU 参照列の族に対する最適ページング・アルゴリズムを考えることは 6.3 節の意味で意味のあることである．この問題に対する解答を与える前に，このモデルの諸性質の内のある面について論じておくことにしよう．

このモデルの利点は，参照間の相関関係が実際に見られるものとあまりかけ離れていないような参照列のパターンを産み出すことができるというモデルの能力から来ている．たとえば，$a_k = 1$ かつ $a_i = 0(i \neq k)$ であるように距離分布を選ぶならば，参照列はループ

$$\omega = k, k-1, \cdots, 2, 1, k, k-1, \cdots, 2, 1, k, k-1, \cdots$$

を構成することになる．もう一つ例を挙げると，参照が距離の小さい方に偏るように距離分布を選ぶならば（たとえば，$a_1 \geq a_2 \geq \cdots \geq a_m$ であるように），参照列は，スタックの先頭に近いページに参照が蝟集する傾向を示すことになり，これによって，局所性(locality)の大事な性質が一つモデル化されることになる．（第7章ではもっと多くのことが局所性に関して述べられる．）反対に，参照が距離の大きい方に偏るような距離分布であるならば，参照列は，参照を

†訳注　いかなる初期スタックが与えられても，ページ番号の付け換えによって $\mathbf{s}_0 = [1, 2, \cdots, n]$ と置けるから，便宜的であり，一般性を失っていない．

多くのページにわたってランダムに撒き散らす傾向を示すことになる．

前述した $\mathbf{s}_0$ の細部指定は(便宜上のものであるので)別とすると，(スタック距離のみを用いて与えられる)本節での定義は，特定のページ番号の指定からは独立している参照列の確率的な記述を意図したものであるということに気付くはずである．(この点は前節で扱った独立参照モデルを想起すると明らかに対照的である．)そこで，LRU 参照列において任意に与えられた参照がページ $i$ に対するものである平衡状態確率は，全ての $i$ について $1/n$ であると考えても少しもおかしくはない．演習問題6.15では，全ての $i$ について $a_i > 0$ を仮定して，事実その通りになることが示される．

**定理 6.7**

LRU ページング・アルゴリズムは，距離分布が下記の条件

$$a_1 \geq a_2 \geq \cdots \geq a_n$$

を満たすならば，常に，この $\{a_i\}$ によって定義される LRU 参照列の族および全ての $m \geq 1$ に対して最適である．

**証明**　LRU 参照列を $r_1r_2\cdots r_t\cdots$ とし，任意に与えられた記憶内容 $S = S_t$ $(|S| = m)$ とスタック $\mathbf{s} = \mathbf{s}_t$ に対して，$r_{t+1}\cdots r_{t+k}$ の処理の間に発生するページ落ちの回数の最小期待値を $C_k(S, \mathbf{s})$ と書く．ページ $x$ の参照とそれに伴うスタック $\mathbf{s}$ の更新の結果新しく得られるスタックを $\mathbf{s}\cdot x$ で表すものとする．まず，次の最適性原理が成立する．ただし，$x$ が $\mathbf{s}$ 内の位置 $i$ にあるとき，$b_x(\mathbf{s}) = a_i$ であると定義する．

$$C_0(S, \mathbf{s}) = 0 \tag{6.7.4}$$

$$C_{k+1}(S, \mathbf{s}) = \sum_{x \in S} b_x(S) C_k(S, \mathbf{s}\cdot x) + \sum_{x \notin S} b_x(\mathbf{s}) \left[1 + \min_{z \in S} C_k(S + x - z, \mathbf{s}\cdot x)\right]$$

定理6.2の方法に倣って，次式が成り立つことを示せば十分である．

$$C_k(S + w', \mathbf{s}) - C_k(S + w, \mathbf{s}) \geq 0 \tag{6.7.5}$$

ただし，$w'$ は，$S + w + w'$ に含まれているページの中では $\mathbf{s}$ で定義される距離が最大のページである．(ここで，この定理よりも若干強い結果を証明しようとしていることを注意しておく．なぜならば，$S$ と $\mathbf{s}$ は任意であることが許されており，必ずしも $r_1\cdots r_t$ の LRU 処理の結果得られるような値であるとは限らないからである．)帰納法を用いて証明を進めよう．

関係式(6.7.5)は，$k=0$ に対しては明らかに成立する．次に，任意の $k \geq 0$ に対して成立すると仮定して，この式が $k+1$ に対しても成立することを示そう．式(6.7.4)を $C_{k+1}(S+w', \mathbf{s}) - C_{k+1}(S+w, \mathbf{s})$ に適用すると，次式が得られる．

$$
\begin{aligned}
&C_{k+1}(S+w', \mathbf{s}) - C_{k+1}(S+w, \mathbf{s}) \\
&= \sum_{x \in S} b_x(\mathbf{s})\,[C_k(S+w', \mathbf{s}\cdot x) - C_k(S+w, \mathbf{s}\cdot x)] + [b_w(\mathbf{s}) - b_{w'}(\mathbf{s})] \\
&\quad + b_w(\mathbf{s})\,[\min_{z \in S+w'} C_k(S+w'+w-z, \mathbf{s}\cdot w) - C_k(S+w, \mathbf{s}\cdot w)] \\
&\quad + b_{w'}(\mathbf{s})\,[C_k(S+w', \mathbf{s}\cdot w') - \min_{z \in S+w} C_k(S+w+w'-z, \mathbf{s}\cdot w')] \\
&\quad + \sum_{x \notin S+w+w'} b_x(\mathbf{s})\,[\min_{z \in S+w'} C_k(S+w'+x-z, \mathbf{s}\cdot x) \\
&\qquad\qquad - \min_{z \in S+w} C_k(S+w+x-z, \mathbf{s}\cdot x)]
\end{aligned}
$$

帰納法の仮定により第1項は非負である．仮定により第2項も非負である．$w'$ は $(S+w+w'$ に含まれるページの中では$)\mathbf{s}\cdot w$ で定義される距離を用いてもやはり距離が最大のページであるから，第3項内の $C_k(S+w'+w-z, \mathbf{s}\cdot w)$ (ただし，$z \in S+w'$)は，帰納法の仮定により $z=w'$ のとき最小(最適)となる．このとき，第3項は零になる．選択(必ずしも最適ではない)$z=w$ によって第4項は零になる．したがって，最小費用を齎す $z$ は第4項を非負にする．最後に，第3項と第4項が非負であることを示した論法を用いて第5項が非負であることを証明するのは容易である．■

もしも $m$ **が固定されている**と考えるならば，定理6.7の証明はもっと弱い条件 $\min\{a_1, \cdots, a_m\} \geq \max\{a_{m+1}, \cdots, a_n\}$ の下で行うことができる．さらに，もしも記憶領域の大きさの系列 $\{m_t\}$ が与えられるならば，それぞれ記憶領域の大きさが一定に保たれるように期間を区切り，定理6.7を個々に適用することにより，記憶領域の大きさが変動する場合でもLRUは最適であるという結論を下すことが可能である．これは，第7章で検討される多重プログラミングと明らかに関わりがある．

最後に，距離分布 $\{a_i\}$ を持つLRU参照列の族に対して次式が成立することを注意しておく．

(6.7.6) $$F(\text{LRU}) = 1 - A_m$$

この結果は初期スタック $\mathbf{s}_0$ に独立であるということが(定理6.4の方法を用いて)容易に確かめられる．さらに，ページ落ち発生の時間間隔の平衡状態分布

はパラメータ $A_m$ を持つ幾何分布であるという結果も式(6.7.6)から得られる.

## 演習問題

**6.1** 次のように動作する**必要時プリページング**(demand prepaging)・アルゴリズムを考察したい. すなわち, 記憶領域中にないページが参照されるとき, しかもそのときにのみ, 好むだけのページ数(参照されるページを必ず含む)を取込むことが許される.

**a.** このような型の必要時プリページング・アルゴリズムを定式化せよ.

**b.** 参照列が予め知られていると仮定して, 搭載するべき余分のページと排除するべきページを決定するための最適アルゴリズムを($B_0$ を拡張して)求めよ.

**c.** 上記の型の実現可能なプリページング・アルゴリズムを二三提案し, それがよいと考えられる理由を述べよ.

**6.2** 本章で LRU および FIFO アルゴリズムに適用されたマルコフ連鎖の方法を用いて, 式(6.6.5)で与えられた $F(A_0)$ に対する結果を証明せよ. $A_0$ アルゴリズムに対して, 状態($i$ 番目の状態であるとしよう)が単一のページ $\{z\} = S_i - L_m$(ただし, $L_m = \{1, \cdots, m-1\}$)で与えられるようなマルコフ連鎖を定義できることに注意しよう.

**6.3** 式(6.6.16)で与えられた $\pi_s$ が式(6.6.12)を満足することの証明を $m$ についての帰納法を使って行え.

**6.4** LFU アルゴリズムに対して拡張問題を解くことにする. より大きい記憶領域に対する排出-充塡系列を計算するために, ページ落ち発生時に記録されなければならない最小量の情報(状態変数)を定義せよ. また, それに対応するアルゴリズムを示せ.

**6.5** $A_0$ が最適であること(定理6.2を見よ)の完全な証明を与えよ.

**6.6** FIFO アルゴリズムの解析のために定義したマルコフ連鎖 $\{\mathbf{s}_j\}$ が既約であることを示せ. 特に, 有限の段階(すなわち, 遷移)——それぞれ非零の遷移確率を持つ——を経て, 与えられた任意の状態$[j_1, \cdots, j_m]$から他の任意の状態$[j'_1, \cdots, j'_m]$へ移行することができることを示せ.

**6.7** $m < m' < 2m-1$ および $n \geq [m' + m/2]$ を満たす $n$ ページの集合と二通りの記憶領域の大きさ $m$ および $m'$ を考える. $K_m[\omega]$ は, 大きさ $m$

の記憶領域(最初は空である)において FIFO アルゴリズムにより参照列 $\omega$ に対して発生するページ落ちの回数を表すものとする．二つの参照列 $\omega_1$, $\omega_2$ をそれぞれ

$$\omega_1 = m,\ m+1,\ \cdots,\ m'-1,\ m',\ 1,\ 2,\ \cdots,\ m-1$$
$$\omega_2 = m,\ 1,\ m+1,\ 2,\ m+2,\ 3,\ \cdots,\ n,\ n-m+1,\ 1,\ n-m+2,\ \cdots,\ m-1,\ n$$

と定義し，$k$ を十分に大きく取るならば，

$$\frac{K_{m'}[\omega_1(\omega_2)^k]}{K_m[\omega_1(\omega_2)^k]}$$

はいくらでも 2 に近付けられることを示せ．この変則的振舞が生じるためには $m$, $m'$, および $n$ に関する上記の条件が必要であることを示せ．$m=4$, $m'=6$, かつ $n=8$ の場合，すなわち，

$$\omega_1 = 4,\ 5,\ 6,\ 1,\ 2,\ 3$$
$$\omega_2 = 4,\ 1,\ 5,\ 2,\ 6,\ 3,\ 7,\ 4,\ 8,\ 5,\ 1,\ 6,\ 2,\ 7,\ 3,\ 8$$

である場合を例に取って検討を始めよ．

**6.8** 本問題および次の二つの問題では，ページ落ち率に対するページの大きさの影響についていくつかのことを調べる[19]．$\omega = \{r_t\}$ は語の番地の系列であるとする．ページの大きさ $2^a (a \geq 0)$ に対応して，$\omega$ の各項から下位の $a$ ビットを削除することにより，$\omega(a) = \{r_t{}^a\}$ が定義される．この定義により，$\omega(a)$ は，$\omega(a+1)$ において参照されるページの半分の大きさのページに対する参照列となり，$\omega(a)$ において参照される各ページは，対応する $\omega(a+1)$ におけるページを半分に分割して得られる二つのページの一方である．ページ $i'$ と $i''$ が元のページ $i$ のそれぞれ半分であるならば，$i'$ と $i''$ は**相棒**(buddies)であるという．

与えられた記憶領域の大きさ $m$ に対し，$F_a(A, \omega)$ は，ページの大きさを $2^a$ としたとき，$\omega$ の上で動作するアルゴリズム $A$ のページ取入れ率(page-entry rate)(すなわち，ページ落ち率)を表すものとする．なお，$m$ は $2^a$ の倍数であると仮定する．まず，$F_a(A, \omega) < F_{a+1}(A, \omega)$ が成り立つような列 $\omega$, および $F_a(A, \omega') > F_{a+1}(A, \omega')$ が成り立つような列 $\omega'$ の具体例をそれぞれ挙げ，それにより，ページの大きさを半分にすることは，参照列によってはページ取入れ率を減少させることもあり，また増

加させることもあるということを示せ.

補助記憶から1ページを取出すのに要する費用は $b+c\,2^a$ であるとする．そうすると，ページング費用の期待値は $C_a=(b+c\,2^a)F_a$ で評価される．ここで，パラメータ $b$ は初動時の探索時間または回転待ち時間を, $c$ は(語)転送速度をそれぞれ表す．さて，$C_a \leq C_{a+1}$ が成り立つのはプログラムがどのような動作をするときか，すなわち，ページの大きさを半分にすると有利になるのはどのような動作をするプログラムの場合であるかを論ぜよ．

**6.9** 直観的にはもっともらしい関係 $F_a(A, \omega) \leq 2\,F_{a+1}(A, \omega)$ が成立しないことを以下で示すことにしよう．

**a.** ページの大きさ $2^a$ に対する次の参照列を考えよう．

$$\omega(a)=\alpha_k=(1'2'\cdots n'n''\cdots 2''1'')^k \quad (k \geq 1)$$

これに対応するページの大きさ $2^{a+1}$ に対する参照列は

$$\omega(a+1)=\beta_k=(12\cdots nn\cdots 21)^k$$

となる．$m$ は $2^{a+1}(n-1)$ 語から成るとする．このとき，次式が成り立つことを示せ．

$$\lim_{k\to\infty}\frac{F_a(\mathrm{LRU},\ \alpha_k)}{F_{a+1}(\mathrm{LRU},\ \beta_k)}=n$$

(記憶領域の初期の状態は空であると仮定せよ.)

**b.** $\omega_1=12\cdots n$, また $\omega_2=21\,n\cdots 3$($\omega_2$ は $\omega_1$ の置換である)とする．$i=1, 2$ について，$\omega'_i$ および $\omega''_i$ をそれぞれページ $1', \cdots, n'$ および $1'', \cdots, n''$ の上で $\omega_i$ と同じように定義する．

次に,

$$\omega(a)=\alpha_k=\omega''_1(\omega'_2\omega''_2)^k$$
$$\omega(a+1)=\beta_k=\omega_1(\omega_2\omega_2)^k$$

を定義する．$m$ は $2^{a+1}(n-1)$ 語から成るとする．このとき，次式が成り立つことを示せ．

$$\lim_{k\to\infty}\frac{F_a(\mathrm{FIFO},\ \alpha_k)}{F_{a+1}(\mathrm{FIFO},\ \beta_k)}=n-1$$

(記憶領域の初期状態は空であると仮定せよ,)

**6.10** $A$ は与えられたスタック・アルゴリズムであるとして，次のアルゴリズム $\bar{A}$ を考えよう．あるページが参照されたとき，それが記憶領域内に存在しないならば，$\bar{A}$ は，そのページとその相棒の両者を取込む[†]と同時に，$A$ が排除するページに対応する(このページを二分割して得られる)相棒の対を排除する．明らかに，$A$ と $\bar{A}$ の両者に対して，記憶領域内には同一の情報が存在することになる．また，$2F_{a+1}(A, \omega) = F_a(\bar{A}, \omega)$ が成り立つこと，$\bar{A}$ は必要時ページング・アルゴリズムではないことに注意しよう．定理6.1によって，$F_a(\bar{A}', \omega) \leq F_a(\bar{A}, \omega)$ であるような必要時ページング・アルゴリズム $\bar{A}'$ が存在する．$\bar{A}'$ の動作の詳細を述べ，たとえ $A$ がスタック・アルゴリズムであっても，$\bar{A}'$ はそうとはいえないことを示せ．

$B_0$ の最適性を用いて次式が成り立つことを示せ．

$$2F_{a+1}(B_0, \omega) = F_a(\bar{B}_0, \omega) \geq F_a(\bar{B}'_0, \omega) \geq F_a(B_0, \omega)$$

ただし，$\bar{B}_0$ と $\bar{B}'_0$ はそれぞれ $\bar{A}$ と $\bar{A}'$ の制約に従って $B_0$ から得られる．すなわち，最適アルゴリズムは，ページの大きさを半分にしてもページ取入れ率が2倍を超過して増大することはないという特性を持つ．

**6.11** 本問題および続く三つの問題では，達成可能な最小のページ落ち率に対する必要時ページング・アルゴリズムのページ落ち率の比の上(下)界を考える．ページ要求(参照)確率 $\beta_1, \beta_2, \cdots, \beta_n$ を持つ6.6節の独立参照モデルを仮定する．全てのページ排除規則の中での最小のページング率(すなわち，ページ落ち率)を $F(\mathrm{MIN})$ で表す．$\beta_1 \geq \beta_2 \geq \cdots \geq \beta_n$ と仮定し，主記憶領域は $m$ 個のページ枠から成るものとする．MIN は，明確に述べると，最初の $m$ ページ $\{1, \cdots, m\}$ を主記憶領域内に保持し続けるアルゴリズムである．したがって，MIN は通常の意味のページング・アルゴリズムではない．$F(\mathrm{MIN})$ については次式が成立する．

$$F(\mathrm{MIN}) = B' = \sum_{i=m+1}^{n} \beta_i$$

さて，次の式が成立することを証明せよ[20]．

$$1 \leq \frac{F(A_0)}{F(\mathrm{MIN})} \leq 1 + \frac{(1 - B')}{1 + (m-1)B'} \leq 2$$

---

†訳注　ページ取入れ回数は2となることに注意せよ．

**6.12**　$F(\mathrm{LRU})$ を次のように表すことができる．すなわち，

$$F(\mathrm{LRU})=\sum_{i=1}^{n}\beta_i p_i\le B'+\sum_{i=1}^{m}\beta_i p_i$$

ただし，$p_i$ は，定常状態においてページ $i$ が主記憶領域内に存在しない確率である．ページ $i(1\le i\le m)$ が主記憶領域内に存在しないためには，次の二つの条件が成り立つことが必要である．すなわち，

1．ページ $i$ の最後の参照以後，$j>m$ であるようなあるページ $j$ が参照されていなければならない．しかも，

2．ページ $i$ は $m-1$ 回の参照の間参照されてはいない．

この観察から，次式が成り立つことを証明せよ [20]．

(6.12.1)
$$p_i\le\left(1-\frac{\beta_i}{\beta_i+B'}\right)(1-\beta_i)^{m-1}$$

したがって，次式が得られることを示せ†．

(6.12.2)
$$\frac{F(\mathrm{LRU})}{F((\mathrm{MIN})}\le 1+\frac{m(1-B')}{1+(m-1)B'}$$

後者の不等式については，一般に関数 $f(x)$ が凸，すなわち，

$$2f\left(\frac{x_1+x_2}{2}\right)\ge f(x_1)+f(x_2)$$

ならば，$\sum_{i=1}^{m}f(\beta_i)$ は $\beta_1=\beta_2=\cdots=\beta_m$ のときに最大となるという事実を利用して証明せよ．

**6.13**　ここでは，与えられた性能を LRU アルゴリズムを用いて保証するのに必要な主記憶領域の大きさに関する上界を求めることを目的とする．$m+k$ ページの大きさの主記憶領域があると仮定する．このとき，ページ $i(1\le i\le m)$ が主記憶領域内に存在しないための必要条件は，ページ $i$ に対する最後の参照以後に $\{m+1,\cdots,n\}$ の中の少なくとも $k+1$ 個の相異なるページが参照されていなければならないことである．このことを示すと共に，この条件が成立する確率は次式より大ではあり得ないこと

---

†訳注　求めたい結果 (6.12.2) を得るためには，式 (6.12.1) の証明から行う必要はなく，式 $p_i\ge\dfrac{B'}{\beta_i+B'}$ の証明を直接に実行することから始めればよいことを特に注意しておく．なお，ヒントとして，$\dfrac{\beta_i}{\beta_i+B'}$ はページ $i$ の最後の参照以後 $j>m$ であるようなどのページも参照されていない確率であること (まず，このことを証明せよ) を述べておく．

を示せ[†].

$$g(\beta_i) = \left(\frac{B'}{\beta_i + B'}\right)^{k+1}$$

次に，次式

$$\max_{\substack{0\le\beta_i\le1\\k\ge1}} \beta_i g(\beta_i) = \frac{B'}{k}\left(1-\frac{1}{k+1}\right)^{k+1}$$

が成り立つことを示し，したがって，次の結果が得られることを示せ[20].

$$\frac{F_{m+k}(\mathrm{LRU})}{F_m(\mathrm{MIN})} \le 1 + \frac{\sum_{i=1}^{m}\beta_i p_i}{B'} \le 1 + \frac{m}{k}\left(1-\frac{1}{k+1}\right)^{k+1}$$

ただし，記号 $F_j(\cdot)$ の添字 $j$ は，ページ落ち率 $F(\cdot)$ に対応する主記憶領域の大きさを明示するためのものである．上式は，$k$ が大きいときには $1 + m/(ke)$ で近似できることを注意しておく．

**6.14** どの $\varepsilon > 0$ に対しても，FIFO アルゴリズムに対して

$$\frac{F_{m+k}(\mathrm{FIFO})}{F_m(\mathrm{MIN})} \ge 1 + \frac{m}{k+1} - \varepsilon$$

が成り立つような確率分布 $\{\beta_1, \cdots, \beta_n\}$ が存在することを示せ． [FIFO に対する 6.6 節の結果を利用せよ．ただし，$\beta_i = (1-B')/m (1 \le i \le m)$，および $\beta_i = B'/(n-m) (i > m)$ を仮定せよ．] 演習問題 6.13 における結果とこの結果を比較し，どのような意味が含まれているかを考察せよ．

**6.15** 参照列生成の LRU スタック・モデル(6.7 節)を考えよう．

**a.** このモデルによって生成される参照列 $r_1 r_2 \cdots r_t \cdots$ は独立な確率変数の系列ではないということを示せ．

**b.** $X_i(j)$ は $i$ 番目の参照直後の LRU スタックにおけるページ $j$ の位置であるとし，マルコフ連鎖 $\{X_i(j)\}$ を定義せよ．ただし，$X_0(j) = j$ とする．また，全ての $i$ に対して $X_i(j) \in \{1, \cdots, n\}$ である．スタックにおけるページ $j$ の位置の定常状態における確率分布を $\{\pi_k\}$ で表す．また，全ての $h$ に対して $a_h = \Pr[D_t = h] > 0$ であると仮定する．

---

[†]訳注 ヒントを述べる．等式 $1 - x^{k+1} = (1-x) + (1-x)x + \cdots + (1-x)x^k$ を適用して $1 - g(\beta_i)$ を展開し，得られた式の意味を考えよ．

連鎖 $\{X_i(j)\}$ の遷移確率行列は二重に確率的(すなわち，どの行，どの列を取ってもその要素の総和は1である)であり，したがって，全ての $k$ に対して $\pi_k = 1/n$ となることを示せ．これは以下のことを意味する．すなわち，長時間走行においては，上記により，どのページも確率的に同等に参照されることになる．その結果，本モデルは，この性質を満たしつつ，6.7節で述べたことにより，参照が局所に密集する現象を取扱う解析手段として有効である．しかしながら，一様でないページ参照† の取扱いに対しては使用できない．

6.16　6.6節で記述された技法を用い，独立参照モデルに対して，次式が成立することを示せ[21]．

$$F(\mathrm{RAND}) = F(\mathrm{FIFO})$$

ただし，RAND は，現在記憶内容から排除ページをランダムに選出するページ排除規則を表す．[明確に述べると，(一杯の状態になっている)記憶領域内に存在する各ページはいずれも排除ページとして選ばれる確率が $1/m$ である．]

6.17　記憶状態 $S_t = S$，および LRU スタック・モデルにより初期スタックを $\mathbf{s} = \mathbf{s}_t$ として生成された参照列 $r_{t+1}\cdots r_{t+k}$ が与えられたとする．このとき，記憶内容 $S$ とスタック $\mathbf{s}$ に対して，$r_{t+1}\cdots r_{t+k}$ を処理するのに要する最小の必要時ページング費用を $C_k(S, \mathbf{s})$ で表す．

スタック $\mathbf{s}$ の最上位 $m$ 個の要素を $S$ で表し，$\mathbf{s}$ から取られた $m$ ページから成る任意の集合を $S'$ としよう．まず，次式が成り立つことを示せ．

$$0 \leq C_k(S', \mathbf{s}) - C_k(S, \mathbf{s}) \leq |S - S'|$$

次に，これを用い，LRU 参照列の上では LRU アルゴリズムのページ落ち確率は漸近的に $1 - A_m$ となることを示せ[式(6.7.6)を見よ]．すなわち，

$$F(\mathrm{LRU}) = \lim_{k\to\infty} \frac{C_k(S', \mathbf{s})}{k} = 1 - A_m$$

---

†訳注　局所性集合というものは‘長時間走行におけるページ参照’が全てのページに対して確率的に同等(一様)であるのではないということによって形成されるのではないことに注意しなければならない．7.2節および7.3節では，(時点 $t$ における)局所性集合 $L_t$ は時刻 $t$ の推移と共にゆっくりと遷り変って行くものとして定義される．

# 参考文献

1. Aho, A. V., P. J. Denning, and J. D. Ullman, "Principles of optimal page replacement." *J. ACM 18*, 1 (Jan. 1971), 80–93.
2. Mattson, R. L., J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies." *IBM Sys. J. 9*, 2 (1970), 78–117.
3. King, W. F., III, "Analysis of paging algorithms." *IFIP Conf. Proc.* Ljubljana, Yugoslavia (Aug. 1971).
4. Coffman, E. G., Jr., and B. Randell, "Performance predictions for extended paged memories." *Acta Informatica 1*, 1 (1971), 1–13.
5. Knuth, D. E., *The Art of Computer Programming* (Vol. 1). Addison-Wesley, Reading, Mass. (1968), 435–455.
6. Purdom, P. W., Jr., and S. M. Stigler, "Statistical properties of the buddy system." *J. ACM 17*, 4 (Oct. 1970), 683–697.
7. Robson, J. M., "An estimate of the store size necessary for dynamic storage allocation." *J. ACM 18*, 3 (July 1971), 416–423.
8. Belady, L. A. "A study of replacement algorithms for virtual storage computers." *IBM Sys. J. 5*, 2 (1966), 78–101.
9. Belady, L. A., R. A. Nelson, and G. S. Shedler, "An anomaly in the space-time characteristics of certain programs running in paging machines." *Comm. ACM 12*, 6 (June 1969), 349–353.
10. Coffman, E. G., Jr., and L. C. Varian, "Further experimental data on the behavior of programs in a paging environment." *Comm. ACM 11*, 7 (July 1968), 471–474.
11. Kuck, D. J., and D. H. Lawrie, "The use and performance of memory hierarchies—A survey," in *Software Engineering* Vol. I (J. T. Tou, ed.). Academic Press, New York (1970), 45–78.
12. Pomeranz, J. E., "Paging with fewest expected replacements." *Proc. IFIPS Congress* (Aug. 1971), 160–162.
13. Coffman, E. G., Jr., and N. D. Jones, "Priority paging algorithms and the extension problem." *Proc. 11th Switching and Automata Theory Symp.* (Oct. 1971).
14. Denning, P. J., J. E. Savage, and J. R. Spirn, "Models for locality in program behavior." Department of Electrical Engineering, Princeton Univ., Princeton, N. J., Computer Science report TR-107 (Apr. 1972).
15. Oden, P. H., and G. S. Shedler, "A model of memory contention in a paging machine." IBM T. J. Watson Research Center report RC-3053, Yorktown

Heights, N.Y. (Sept. 1970).

16. SHEDLER, G. S., and C. TUNG., "Locality in page reference strings." *SIAM J. Comput. 1*, 3 (Sept. 1972), 218–241.

17. SHEMER, J. E., and S. C. GUPTA, On the design of Bayesian storage allocation algorithms for paging and segmentation." *IEEE Trans. Comp. C-18*, 7 (July 1969), 644–651.

18. SHEMER, J. E., and B. SHIPPEY, "Statistical analysis of paged and segmented computer systems." *IEEE Trans. Comp. EC-15*, 6 (Dec. 1966), 855–863.

19. HATFIELD, D. J., "Experiments on page size, program access patterns, and virtual memory performance." *IBM J. Res. Develop. 16*, 1 (1972), 58–66.

20. FRANASZEK, P. A., and T. J. WAGNER., "Some distribution-free aspects of paging algorithm performance." Technical report, IBM Research Center, Yorktown Heights, N.Y. (Apr. 1972). To appear in *J. ACM.*

21. GELENBE, E. "Random partially pre-loaded page replacement algorithms." Technical report, Electrical and Computer Engineering Dept., Uni. of Michigan. Ann Arbor (Nov. 1971).

# 7 多重プログラミング下の記憶領域管理

## 7.1 はじめに

第6章における諸結果は，大きさの固定された記憶空間におけるページング・アルゴリズムの振舞だけに専ら係わりがあった．また，独立参照モデルを除いて，その全ての結果は，任意ではあるものの明細を指定されている参照列に関するものであった．実際の多重プログラミングの下での状況は，往々にして，一つのプログラムに対して大きさが一定の動作空間を保証することをしないし，プログラム参照列についての知識を予め持てるというようにはなっていない．そこで，上記の二つの制約条件を取り払う必要が生じる．

第1章で注意したように，多重プログラミングにおいて関心が持たれる基本的問題は，主記憶空間 $M$ を活動中のタスク(active task)の間で分割することである．明確に述べると，各時点 $t$ で記憶領域の**分割**(partition)

(7.1.1) $$\mathbf{Z}(t) = (Z_1(t), \cdots, Z_n(t))$$

が存在するということになる．ここで，$Z_i(t)$ は，活動中のタスク $i$ に属するページの集合(またはブロックともいう)を表していて，これらは次式を満たすものとする†.

(7.1.2) $$\bigcup_{i=1}^{n} Z_i(t) \subseteq M,\ Z_i(t) \cap Z_j(t) = \phi \quad (i \neq j)$$

式(7.1.1)に関しそれだけでいくつかの問題が浮かび上る．すなわち，1)分割 $\mathbf{Z}(t)$ は固定であるべきか，あるいは可変であるべきか．2) $\mathbf{Z}(t)$ の各ブロックはどのように決定されるべきか．3) $Z_i(t)$ の大きさがある期間一定に保たれる

† 共有ページ[$Z_i(t)$ と $Z_j(t)$ の共通部分]に関する問題は，ここでは考えないことにする．

ならば，$Z_i(t)$ へのページ取込み時と排除ページを決定するためにはどのような方策を用いるべきか．後に分るように，これらの疑問に対して答を得る前に，プログラムの記憶領域要求の本質的な(機械に依存しない)尺度を定義する必要がある．本章で用いる尺度は，プログラムの**作業集合**(working set)という名で知られているものである．

残念なことに，スタック・アルゴリズムの技法(第6章)は，多重プログラミングに向かってどんな有益な一般化も齎さない．その第一の理由は次の通りである．もしも記憶空間が変動することを許すならば，一般に，優先権アルゴリズムは，プログラムが $m$ ページを割当てられているときにはいつでもスタックの先頭から上位 $m$ 個の要素が記憶領域内にあるという意味でのスタック・アルゴリズムではなくなるからである．例として，この状況の下で，3ページから成り，下表のように動作するプログラムを考えてみよう．

| | 時間 | |
|---|---|---|
| | $t$ | $t+1$ |
| 記憶空間 | 3 | 2 |
| 参照ページ | — | 2 |
| 優先順位リスト | (1, 2, 3) | — |
| スタック | (3, 2, 1) | (2, 3, 1) |
| 記憶状態 | {1, 2, 3} | {1, 2} |

記憶状態 $S_{t+1}$ は，最低の優先権を持つページを $S_t$ から削除することによって得られる．明らかに，$S_{t+1}$ はスタック $\mathbf{s}(t+1)$ の先頭から上位2個の要素から成らない．記憶空間の割当てに変動があるという条件の下でもスタック・アルゴリズムとなる唯一の優先権ページング・アルゴリズムは LRU アルゴリズムである．なぜならば，その優先順位リストは常にスタックに等しいからである(6.4.2節を参照せよ)．

スタック・アルゴリズムに関する諸結果は参照列の顕在的な諸特性に依存してはいない，換言すれば，それを利用していないということが，スタック・アルゴリズムの多重プログラミングに向かっての有効な一般化が齎されない第二の理由である．これらの理由から，**局所性**(locality)――多くの実際の参照列によってその存在が示されている――の持つ諸性質について議論することから多重プログラミングの検討を始め，この特性を利用することによって管理アルゴリズムを最適に近く動作させるにはどのような設計ができるかを研究する必

要がある．ここでは，プログラムのモデルそのもの(すなわち，プログラムが参照列を生成する機構のモデル)を議論する意図はない．本章の目的は，どのようなモデルを用いて参照列の生成を記述しようとも，そのことには関わりなく，局所性を満足する任意の参照列に適用する際に現れる多重プログラミング下の記憶領域管理手続きの諸性質を顕示することである．

## 7.2 局 所 性

非形式的に述べるならば，局所性の特質は，プログラムがどんな時間間隔内においてもそのページ集合のある部分集合に偏った参照を行う傾向があること，さらに，そのような偏好ページの集合の構成要素はゆっくりと変化する傾向があるということを意味する．この性質は，プログラムの振舞の実験的研究[1—8]において繰り返し観察されてきた．そこで，基本的な性質としてプログラムの振舞についてのここでの仮説にこれを含めるのは自然であるように思われる．

局所性の性質は，次の三つの文に要約することができる．

**L1.** どんな時間間隔の間でも，プログラムは，どのページにも一様にというのではなく一部のページに参照を集中する．

**L2.** すぐ前の過去とすぐ後の未来の参照パターンの間の相関は高いという傾向があり，互いに重ならない部分参照列の間の相関は，それらの間の距離が増大するにつれて零に近づく．

**L3.** 時間の関数として見ると，与えられたページが参照される頻度はゆっくりと変化する傾向がある．すなわち，準定常的(quasi-stationary)である．

局所性のこれら三つの性質の中で，L3は結果を適用する際に最も厳しい制限を課することになる．これを含めたのは次の三つの理由からである．第一に，有効な解析技法の大部分が仮定L3を必要とするモデルに限定されているからである．第二に，非定常的振舞のある一面の状況は定常モデルによって，あるいは定常モデルの系列によって近似することが可能であるからである．第三に，プログラムがL1—L3の通りになる度合は，適切に訓練されたプログラマによって[2, 7, 9]，あるいはページ間に命令コードとデータ・コードを適切に配分することによって[10]強められ得るという少なからぬ証拠があるからである．それゆえ，原理的には，局所性原理を満足するプログラムを作り上げるようにプログラマとコンパイラに影響をおよぼすことが可能である．

局所性原理は，実際から看取された少なくとも三つの現象の抽象である．第一に，プログラムには，逐次実行型と繰返し型の制御構造が頻用される傾向があり，短時間ある与えられた一群のページに対して参照が密集することになる．第二に，プログラマというものはかなりの期間にわたって大きな問題の小部分に彼等の注意を集中する傾向があり，また，アルゴリズムの方式とデータ編成の単純な変更がこの効果を飛躍的に増大させることもある [2, 7, 10]．第三に，プログラムは主記憶領域内にページ集合の一部を置くだけで効率よく走行し得ることが経験上分っている [7, 11, 12, 14]．

局所性の性質は非常に望ましいものであることを上記は示唆している．というのは，それらが顕著であればある程，効率のよい実行のためにプログラムが必要とする主記憶領域量は少なくて済みそうであるし，直前の過去の参照列の観察に基づくプログラムの記憶領域必要量の予測をさらに高い信頼性をもって行うことが可能になるからである．たとえば，**作業集合モデル**(working set model)(すぐ後で論じられる)——これは局所性の諸仮定に直接基礎を置いている——は，実際に有効に機能し得ることが知られている [6, 14]．それが未来の記憶領域要求の信頼し得る予測子(predictor)となる度合は，L2とL3への忠実さが増すに従って大きなものになる．

## 7.3　作業集合モデル[3, 15, 16]

ページの集合が $N=\{1, \cdots, n\}$ であるような $n$ ページのプログラムを考えよう．ここでは，ページの大きさ，また番地空間の各語をページの間に分配する仕方のいずれにも関心を持たない．プログラムの作業集合とは，直観的には，ページ落ち率を与えられた水準以下に保つために主記憶領域上に搭載しておかなければならない $N$ の最小部分集合のことである．

### 7.3.1　参照列に関する諸仮定

プログラムの(ページ集合 $N$ の上の)参照列 $r_1 r_2 \cdots r_t \cdots$ は，確率変数の系列，すなわち，離散的確率過程と見なすことができる．関心が持たれる基本的な遷移確率は

$$(7.3.1)\qquad g_{ij}(t, x)=\Pr[r_{t+x}=j|r_t=i]\quad (i, j\in N, t\geq 1, x\geq 1)$$

である．

以下の議論においては，参照列に関して三つの仮定を置くことにする．第一に，遷移確率は**定常的**(stationary)であると仮定する．すなわち，$g_{ij}(t, x)$ は

区間$(t, t+x)$の起点に依存しない．(これは局所性の性質 L 3 のより限定された言い方である．)特に興味があるのは定常的な**参照間隔**(interreference)**密度**と**分布**であり，それぞれ，$g_{ij}(t, x)$ から次のように定められる．

$$(7.3.2) \qquad f_i(x) = \begin{cases} g_{ii}(t, 1) & (x = 1) \\ g_{ii}(t, x) - \sum_{k=1}^{x-1} f_i(k) g_{ii}(t+k, x-k) & (x > 1) \end{cases}$$

$$(7.3.3) \qquad F_i(x) = \sum_{k=1}^{x} f_i(k)$$

$f_i(x)$ は，同一ページ $i$ への引き続く二つの参照の間隔が値 $x$ の長さになる確率であると解釈されることに注意しよう．これは，待ち行列システムの解析において以前に使用した到着間隔分布に類似するものである．

三つの仮定の中で，定常性は実際問題として結果の適用性に最も強い制限を課する．これらの制限は次の考察によって明瞭にすることができる．局所性の仮定に従うと，プログラムというものは，いくつもの**局所性集合**(locality)——一つの局所性集合は $N$ のある部分集合であり，**現在(の)局所性集合**(current locality)のページが高い確率で参照される——の間の経時遷移を惹き起こすものであると考えることができよう．明確にするため，プログラムは，参照 $r_1r_2\cdots r_t\cdots$ を発生しながら，局所性集合の系列 $L_0L_1\cdots L_t\cdots(L_t \leq N)$を通過し，しかも高い確率で $L_t = L_{t-1}$ が成り立つと仮定する．$t_1 \leq t < t_2$ に対して $L_t = L$ であるならば，全ての $t(t_1 \leq t < t_2)$に対して $r_t \in L$ である．そこで，参照列 $\omega$ を $\omega = \omega_1\omega_2\cdots\omega_i\cdots$ と置いて，$\omega_i$ が系列 $L_0L_1\cdots L_t\cdots$ における $i$ 番目の異なる局所性集合†の上で生成される参照列であるようにすることができる．

ところで，いかなる非定常性が $\omega$ に内在しようとも，それが局所性(の性質 L 3)にまずは従うことを期待してよいであろう．すなわち，$\omega_i$ は定常的な確率機構によって生成されるものと見ることができる．したがって，定常性の仮定は要するにプログラムの局所性集合のどれか一つへの局限であるということになる．参照列 $\omega = \omega_1\omega_2\cdots\omega_i\cdots$ は，それぞれが定常性の仮定に従うモデルの系列——系列内の $i$ 番目のモデルは部分列 $\omega_i$ に対処するために用いられる——によって近似できるということに注意しよう．もし，局所性集合間の遷移を多く含むような与えられた参照列の断片にここでの結果を適用したとするならば，その結果は，それが多くの局所性集合の上で平均化されたプログラム

† **訳注** 直前の $L_{t-1}$ とは異なる $L_t$ から成る系列 $L_{t_1}L_{t_2}\cdots L_{t_i}\cdots$ (ただし，$L_{t_1}=L_0$)($L_{t_i} \neq L_{t_{i+1}}$ がもちろん成り立つ)の $i$ 番目の局所性集合 $L_{t_i}$ を指す．

の振舞に関するものであるということから——多数の局所性集合の平均的振舞は任意に与えられた局所性集合を必ずしも代表するものではない——，意味あるものでなくなるであろう．言葉を換えていうと，ここでの結果は参照列の中で局所的に適用可能であるが，必ずしも全域にではない．ここでの主な関心は，記憶領域管理に使える適応性に富んだ評価尺度として，作業集合モデルの振舞をまず理解することであるから，この制約は，測定間隔が局所性集合間遷移時間に同等かより小である限り決して苛酷なものではないということになる．以上の観念が局所性モデルの基盤を形作っているが[17]，そのうちのある面は演習問題でさらに考察される．

参照列に関する第二の仮定は，各ページが**再帰的**(recurrent)，すなわち，

(7.3.4)
$$\sum_{x\geq 1} f_i(x) = 1$$

であり，したがって，任意のページのどの参照に対しても確率1でそれが再び参照されるということである．非再帰的(過渡的)ページを包含するように拡張するのは簡単であるから，これは考えないことにする．ページ $i$ の**平均参照間隔**(mean interreference interval)は次式のように定義される．

(7.3.5)
$$\bar{x}_i = \sum_{x\geq 1} x f_i(x)$$

上記の仮定は，$r_1 r_2 \cdots r_t \cdots$ の中でページ $i$ を参照する時点の部分系列が再生過程(renewal process)であることを保証するから，ページ $i$ 参照の長時間走行確率を次のように定義することができる†．

(7.3.6)
$$\beta_i = \frac{1}{\bar{x}_i}$$

**全参照間隔**(overall interreference)**分布**および**密度**を，それぞれ次のように定義することにする．

(7.3.7)
$$F(x) = \sum_{i \in N} \beta_i F_i(x)$$

(7.3.8)
$$f(x) = F(x) - F(x-1) \quad (x \geq 1)$$

ただし，$F(0) \equiv 0$．ここでの仮定と式(7.3.6)に従えば，平均全参照間隔は次式のようになる．

† 第6章での議論と同じように，長さ $t$ の長い系列を考えると，ページ $i$ の参照数の期待値は近似的に $t\beta_i$ であり，そのうちの二つの間の平均距離は近似的に $t/t\beta_i$ である．漸近的に式(7.3.6)が成り立つ．

(7.3.9) $$\bar{x} = \sum_{i \in N} \beta_i \bar{x}_i = n$$

参照列についての第三の仮定は，**漸近的無相関性**(asymptotic uncorrelation)，すなわち，$r_t$ と $r_{t+x}$ は極限 $x \to \infty$ において相関関係が無くなることである．この仮定は大概の物理的過程に当てはまる．すなわち，これは，現在についての知識は遠い未来を予測するのにほとんど役立たないということを言明している．この仮定は，(個々の参照列の上での)時間平均(time average)は対応する空間平均(ensemble average)に収束するというエルゴード性が成り立つことを保証する．この仮定は局所性の性質 L2 の言い換えである．

### 7.3.2 諸定義

与えられた時点におけるプログラムの作業集合は，理想的には，その時点での現在局所性集合であるべきである．未来の参照列またはプログラムの局所性集合に関する予知情報を欠いたままで，現在局所性集合を見積ることが必要である．これを行う一つの可能な方法は，時刻 $t$ での優先順位リスト $\mathbf{p}(t)$ と整数 $k(t)$ を定義し，それを用いて作業集合を $\mathbf{p}(t)$ の先頭からの $k(t)$ 個の要素であると定義することである．解析し易くしかも実際上実行可能であることが明証されているその一つの特殊な場合は，作業集合を最も最近に参照されたページの集合であると定義することである．

時刻 $t$ におけるプログラムの作業集合 $W(t, T)$ は，時間間隔 $[t-T+1, t]$ において参照されるページ，すなわち，$r_{t-T+1}\cdots r_t$ の中の相異なるページの集合であると定義する．$t<T$ ならば，$W(t, T)$ には $r_1r_2\cdots r_t$ の中の相異なるページが含まれ，$t \leq 0$ ならば，$W(t, T)$ は空である．$W(t, T)$ は参照列の最後の $T$ 回の参照を包容する**窓**(window)の内容であると見ることができるので，パラメータ $T$ は**窓の大きさ**(window size)と呼ばれる．もしも $T$ を下記のように選び得るならば，この定義は現在局所性集合の信頼し得る評価尺度を提供することになる．すなわち，$T$ は

1. 現在局所性集合のページが作業集合から欠落している確率が微小となるように十分に大きく，かつ，
2. 2回以上の局所性集合間遷移が窓に含まれている確率が微小となるように十分に小さく選ばれる．

記憶領域管理の**作業集合原理**(working set principle)は，プログラムの作業集合が主記憶領域内にあるときに限りそのプログラムは活動中(すなわち，プ

ロセッサのサービスを受ける資格を持っている)であることができるということを主張する．排除ページ選択規則について述べると，それは，活動中のプログラムの作業集合内に含まれているページが排除の候補になることはできないということを主張する．特に，多重プログラミング下の記憶領域が作業集合のページで全域を満たされているならば，除去候補のページを創り出すためにはプログラムの中のどれかを不活性化(活動中でなく)する必要が生じる．記憶領域管理のこの原理は，主記憶領域に過重な負担を掛けようとしたことから起こるシステム性能の急落，すなわち，**スラッシング** (thrashing) を十分に予防する [4, 5]．

局所性の特性 **L 1—L 3** が成り立てば成り立つ程，ますます，上記に示した二つの必要事項が満足されるように $T$ を選択し得るように見える．すなわち，作業集合 $W(t, T)$ は現在局所性集合のいよいよ信頼できる見積りとなり得るように思われる．さらに言えば，プログラムの局所性の特性が良性であればある程，現在局所性集合を保持する残り時間の期待値は長くなる．この場合，局所性集合に属するページはそうでないページよりも短い前向き距離(すなわち，次の参照までの時間)を持つ傾向があるということになり，したがって，作業集合原理によって記憶領域を管理することは最適性(第6章で論じたように最長の前向き距離を持つページを排除すること)に向かう傾向を有するということになる．作業集合方策が最適かあるいは極めてそれに近いものとなるような事例は演習問題で論じることにする．

**作業集合の大きさ** (working set size) $w(t, T)$ は，$W(t, T)$ 内のページ数であると定義する．作業集合による記憶領域管理の研究においては，多くの場合，$w(t, T)$ の定常的振舞の特徴付けが行えれば，それで十分である．作業集合は，参照列の時刻 $t$ における LRU スタックの先頭からの上位 $w(t, T)$ 個の要素であると解することができる．そこで，後には作業集合と LRU ページング・アルゴリズムの関係の問題に立戻ることにしよう．

$h(z, T)$ は，定常状態における作業集合の大きさの確率塊関数を表すものとする．すなわち，

(7.3.10)
$$h(z, T) = \lim_{t\to\infty} \Pr[w(t, T) = z]$$

[7.3.4 節で示されるように，$h(z, T)$ は一般の条件の下で正規分布によって近似することができる.] **平均の作業集合の大きさ** (mean working set size) は次のように定義する．

(7.3.11) $$s(T) = \lim_{t\to\infty} E[w(t,\ T)] = \sum_{z=1}^{n} zh(z,\ T)$$

ところで，作業集合の定義は参照列を事後に観察することを基礎としているので，それまで参照されていないページの参照を予測するためにこのモデルを使うことはできない．しかしながら，以前に参照されたページが再び参照される確率を評価するためにはこれを用いることができる．この目的のために，長時間走行における**ページ不在確率**(missing-page probability)$m(T)$を次のように定義する．

(7.3.12) $$m(T) = \lim_{t\to\infty} \Pr[r_{t+1} \notin W(t,\ T)]$$

記憶領域管理の作業集合原理を用いたシステムでは，$m(T)$ は単にプログラムが経験するページ落ちの確率に対して一つの上界になるだけである．というのは，ページは，作業集合からははみ出るけれども，主記憶領域からは排除されないで残留し，その後再び参照される可能性があるからである．$m(T)$ のもう一通りの定義が二値の確率変数を次のように定義することによって定式化される．

(7.3.13) $$\Delta(t,\ T) = \begin{cases} 1 & (r_{t+1} \notin W(t,\ T)\ \text{のとき}) \\ 0 & (\text{そうでないとき}) \end{cases}$$

これを用いて

(7.3.14) $$m(T) = \lim_{t\to\infty} E[\Delta(t,\ T)]$$

### 7.3.3 諸性質

**図 7.3-1** を検討すると，次式の成り立つことが明らかになる．

(7.3.15) $$w(t,\ T+1) = w(t,\ T) + \Delta(t,\ T)$$

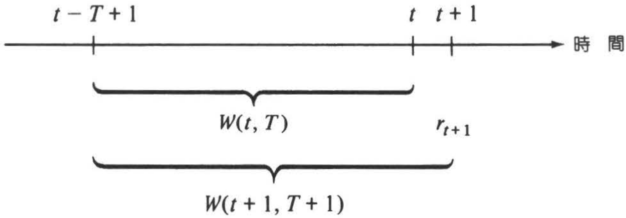



**図 7.3-1** 作業集合の包含特性

式(7.3.15)の両辺について期待値およびその極限を取ると，$s(T+1)=s(T)+m(T)$ という結果が得られる．したがって，次の二つの式を得る．

(7.3.16) $$s(T+1)-s(T)=m(T)$$

(7.3.17) $$s(T)=\sum_{x=0}^{T-1}m(x)$$

これらの式は，$m(T)$ を曲線 $s(T)$ の $T$ における‘勾配’であると見なすことができるということを示唆している．

**例：図7.3-2(a)**は，与えられた20参照から成る列と6通りの $T$ の値に対して作業集合をそれぞれ列挙している．与えられた$(t,\ T)$に対して記入された項は，後向き距離の増加順に従って下向きに縦に並べられた $W(t,\ T)$ の構成要素を示す数字の列になっている．$\Delta(t,\ T)=1$ であるとき，$(t,\ T)$に対する記入項には $\Delta$ 印が付されている．量 $s(T)$ と $m(T)$ はこの参照列の上での時間平均として計算されており，その結果がそれぞれ示されている．**図7.3-2** **(b)**と**(c)**は，$s(T)$ と $m(T)$ をそれぞれ作図したものである．式(7.3.16)は無限長の参照列の極限に基づいているために等式となっているけれども，ここでは，関係式 $s(T+1)-s(T)\leq m(T)$ が成立し，等式とはならないことに注意しよう．長さ $k$ の参照列に対する $s(T+1)-s(T)$ と$m(T)$ の間の誤差は，演習問題7.3に見られるように $\mathcal{O}(1/k)$† である．

二値の確率変数 $\alpha_i(t,\ T)$ を，ページ $i$ が $W(t,\ T)$ 内にあるとき，かつそのときに限り1であると定義する．このとき，次式が成り立つ．

(7.3.18) $$w(t,\ T)=\sum_{i\in N}\alpha_i(t,\ T)$$

平衡状態における平均値は，次式によって与えられる．

$$s(T)=\lim_{t\to\infty}E[w(t,\ T)]=\lim_{t\to\infty}\sum_{i\in N}E[\alpha_i(t,\ T)]$$

$\Pr[\alpha_i(T)=1]=\lim_{t\to\infty}\Pr[\alpha_i(t,\ T)=1]$ と定義すると，次式のようになる．

(7.3.19) $$s(T)=\sum_{i\in N}\Pr[\alpha_i(T)=1]$$

$\Pr[\alpha_i(T)=1]$ を求めるために，これを，平衡状態においてランダムに選んだページ参照 $r_t$ が，経過した時間(参照数)が $T$ を超えていないようなページ

†**訳注**　$\mathcal{O}(1/k)$ は，(たかだか)$1/k$ の位数を持つ関数であることを示す Landau の記号である．

| T | | | | | | | | | | | | | | | | | | | | | s(T) | m(T) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| T | 1 | 2 | 3 | 4 | 5 | 2 | 1 | 3 | 3 | 2 | 3 | 4 | 5 | 4 | 5 | 1 | 1 | 3 | 2 | 5 | s(T) | m(T) |
| 1 | 1 | 2 | 3 | 4 | 5 | 2 | 1 | 3 | 3 | 2 | 3 | 4 | 5 | 4 | 5 | 1 | 1 | 3 | 2 | 5 | 1.00 | 0.90 |
| | Δ | Δ | Δ | Δ | Δ | Δ | Δ | Δ | | Δ | Δ | Δ | Δ | Δ | Δ | Δ | | Δ | Δ | Δ | | |
| 2 | 1 | 2 | 3 | 4 | 5 | 2 | 1 | 3 | 3 | 2 | 3 | 4 | 5 | 4 | 5 | 1 | 1 | 3 | 2 | 5 | 1.85 | 0.75 |
| | | 1 | 2 | 3 | 4 | 5 | 2 | 1 | | 3 | 2 | 3 | 4 | 5 | 4 | 5 | | 1 | 3 | 2 | | |
| | Δ | Δ | Δ | Δ | Δ | Δ | Δ | Δ | | Δ | | Δ | Δ | | | Δ | | Δ | Δ | Δ | | |
| 3 | 1 | 2 | 3 | 4 | 5 | 2 | 1 | 3 | 3 | 2 | 3 | 4 | 5 | 4 | 5 | 1 | 1 | 3 | 2 | 5 | 2.50 | 0.75 |
| | | 1 | 2 | 3 | 4 | 5 | 2 | 1 | 1 | 3 | 2 | 3 | 4 | 5 | 4 | 5 | 5 | 1 | 3 | 2 | | |
| | | | 1 | 2 | 3 | 4 | 5 | 2 | | | | 2 | 3 | | | 4 | | | 1 | 3 | | |
| | Δ | Δ | Δ | Δ | Δ | Δ | Δ | Δ | | Δ | | Δ | Δ | | | Δ | | Δ | Δ | Δ | | |
| 4 | 1 | 2 | 3 | 4 | 5 | 2 | 1 | 3 | 3 | 2 | 3 | 4 | 5 | 4 | 5 | 1 | 1 | 3 | 2 | 5 | 3.10 | 0.65 |
| | | 1 | 2 | 3 | 4 | 5 | 2 | 1 | 1 | 3 | 2 | 3 | 4 | 5 | 4 | 5 | 5 | 1 | 3 | 2 | | |
| | | | 1 | 2 | 3 | 4 | 5 | 2 | 2 | 1 | | 2 | 3 | 3 | | 4 | 4 | 5 | 1 | 3 | | |
| | | | | 1 | 2 | 3 | 4 | 5 | | | | | 2 | | | | | | | 1 | | |
| | Δ | Δ | Δ | Δ | Δ | | Δ | Δ | | | | Δ | Δ | | | Δ | | Δ | Δ | Δ | | |
| 5 | 1 | 2 | 3 | 4 | 5 | 2 | 1 | 3 | 3 | 2 | 3 | 4 | 5 | 4 | 5 | 1 | 1 | 3 | 2 | 5 | 3.55 | 0.55 |
| | | 1 | 2 | 3 | 4 | 5 | 2 | 1 | 1 | 3 | 2 | 3 | 4 | 5 | 4 | 5 | 5 | 1 | 3 | 2 | | |
| | | | 1 | 2 | 3 | 4 | 5 | 2 | 2 | 1 | 1 | 2 | 3 | 3 | 3 | 4 | 4 | 5 | 1 | 3 | | |
| | | | | 1 | 2 | 3 | 4 | 4 | 4 | | | | 2 | 2 | | | | 4 | 5 | 1 | | |
| | Δ | Δ | Δ | Δ | Δ | Δ | | Δ | | | | Δ | Δ | | | Δ | | Δ | Δ | | | |
| 6 | 1 | 2 | 3 | 4 | 5 | 2 | 1 | 3 | 3 | 2 | 3 | 4 | 5 | 4 | 5 | 1 | 1 | 3 | 2 | 5 | 3.90 | 0.50 |
| | | 1 | 2 | 3 | 4 | 5 | 2 | 1 | 1 | 3 | 2 | 3 | 4 | 5 | 4 | 5 | 5 | 1 | 3 | 2 | | |
| | | | 1 | 2 | 3 | 4 | 5 | 2 | 2 | 1 | 1 | 2 | 3 | 3 | 3 | 4 | 4 | 5 | 1 | 3 | | |
| | | | | 1 | 2 | 3 | 4 | 5 | 5 | 5 | | 1 | 2 | 2 | 2 | 3 | | 4 | 5 | 1 | | |
| | | | | | 1 | 1 | 3 | 4 | 4 | | | | | | | | | | 4 | | | |
| | Δ | Δ | Δ | Δ | Δ | | | | | | | Δ | Δ | | | Δ | | Δ | Δ | | | |

(a)

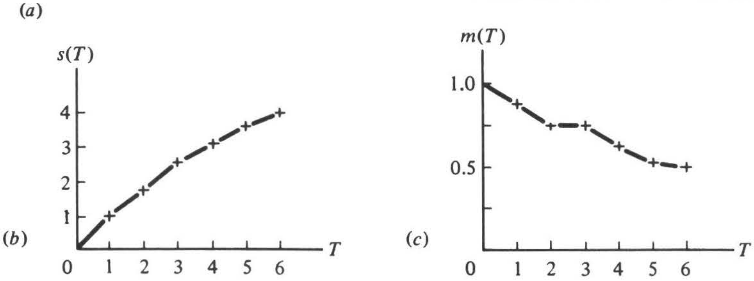



図 7.3-2　例

$i$ の参照間の間隔内に入っている確率と同一視しよう．これは，補遺Bに述べられるいわゆる後退再帰時間(backward recurrence time)の問題である．そこで言及されているように，後退再帰時間の分布と前進再帰時間のそれとは同一である．明確に示すと，式(B.7)から，後退再帰時間の定常分布(4.2.3節と演習問題4.12をも参照せよ)が次の通り得られる．

$$R(T)=\frac{1}{\bar{x}}\int_0^T [1-G(x)]\,dx \tag{7.3.20}$$

ただし，$G(x)$ は引き続く事象(再生)間の時間間隔の分布関数であり，$\bar{x}$ はその1次モーメントである．離散的な当面の場合に対しては，$\bar{x}_i$ で $\bar{x}$ を，$F_i(x)$ で $G(x)$ を，式(7.3.20)内の積分を総和でそれぞれ置き換えればよい．したがって，$\Pr[\alpha_i(T)=1]$ は次のように書ける．

$$\Pr[\alpha_i(T)=1]=\frac{1}{\bar{x}_i}\sum_{x=0}^{T-1}[1-F_i(x)] \tag{7.3.21}$$

(この式は，補遺Bと全く同じ手順を踏むことにより直接に導出することもできる．)したがって，

$$s(T)=\sum_{i\in N}\Pr[\alpha_i(T)=1]=\sum_{i\in N}\sum_{x=0}^{T-1}\frac{1-F_i(x)}{\bar{x}_i}$$

総和の順序を逆にし，式(7.3.6)と(7.3.7)を適用すると，次のようになる．

$$s(T)=\sum_{x=0}^{T-1}(1-F(x)) \tag{7.3.22}$$

さらに，式(7.3.17)を用いると，次式が得られる．

$$m(T)=1-F(T) \tag{7.3.23}$$

以上の結果の重要性は，これによって参照間隔分布，ページ不在確率，および作業集合の大きさの平均値の間の関係が確立したという点にある．

$s(T)$ および $m(T)$ の特性は図7.3-3と7.3-4に要約されている．曲線 $s(T)$ は $\min\{n, T\}$ によって上から押さえられており，しかも，$T$ に関して非減少，かつ，下に凹，すなわち，

$$\frac{s(T-1)+s(T+1)}{2}\leq s(T) \tag{7.3.24}$$

である．ただし，式(7.3.24)は $m(T)$ が非増加的であるという事実から得られる．

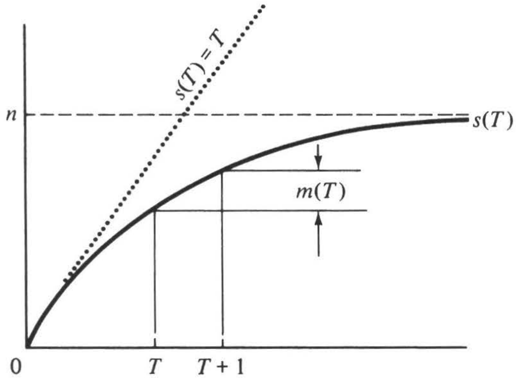



**図 7.3-3** 作業集合の大きさ

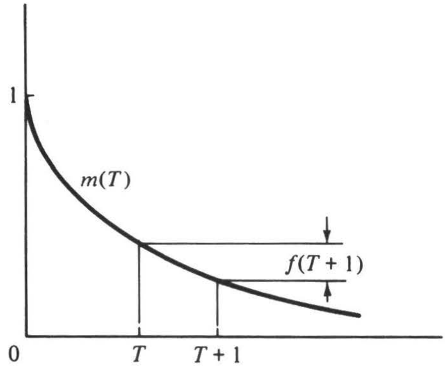



**図 7.3-4** ページ不在確率

### 7.3.4 作業集合の大きさの分布

一般的な条件の下では，作業集合の大きさの分布は近似的に正規分布であること，すなわち，確率過程 $w(t, T)$ は定常正規過程であるということを以前に言及した．局所性の諸条件，取り分け漸近的無相関性の条件は十分にこのことを保証している．正規(ガウス)過程の仮定は，作業集合の大きさの密度関数 $h(z, T)$ が正規密度

$$h(z, T) = \frac{1}{\sqrt{2\pi\sigma^2(T)}} e^{-(z-s(T))^2/2\sigma^2(T)} \tag{7.3.25}$$

によって与えられることを意味する．ただし，平均は

$$s(T)=\int_{-\infty}^{\infty} zh(z,\ T)dz \tag{7.3.26}$$

であり，分散は

$$\sigma^2(T)=\int_{-\infty}^{\infty} (z-s(T))^2 h(z,\ T)dz \tag{7.3.27}$$

である．一般に，$n$ 個の時点 $t_1, \cdots, t_n$ の任意集合に対して，$w(t_1,\ T), \cdots, w(t_n,\ T)$ の結合分布(joint distribution)は結合正規分布となる．そこで，正規過程の諸性質を取扱っている数多くの文献を一覧すると，正規性の仮定は決定的に重要であって，システム設計において大きな利点が生じるように利用できるということが知られる．そうとすれば，正規近似が妥当となる条件を調べることは大変有益である．

$w(t,\ T)$ が定常的であると仮定することは，以下の議論において本質的ではない．準定常性(quasi-stationarity)を仮定すればよい．式(7.3.25)の平均 $s(T)$ と分散 $\sigma^2(T)$ を時間の経過と共にゆっくりと変化する平均 $s(t,\ T)$ と分散 $\sigma^2(t,\ T)$ で置換えることによって良い近似を得ることができる．

正規近似によって直ちに生じる偏差の要因として現実からの二つの逸脱がある．すなわち，この分布が作業集合の大きさの正の値に対してだけでなく，負の値に対しても定義されているということと，作業集合の大きさは離散的であるのに，分布は連続的であるということである．第一の逸脱による偏差の大きさは，関心の対象となる大概のパラメータ値 $s(T)$ および $\sigma(T)$ に対しては無視し得る程である．第二の逸脱による偏差は，作業集合の大きさが十分に数多くの値の上を変化するようなプログラムを取扱うことにより微小な程度に押さえられよう．局所性のモデルによって生成された参照列の上の作業集合の大きさに関する実験的研究によれば，正規近似が極めて良好であり得るということが示されている [18]．

正規近似が妥当であると仮定するならば，それを完全に特徴付けるために平均 $s(T)$ と分散 $\sigma^2(T)$ を知ることが必要となる．これまでに展開してきた解析の道具によって $s(T)$ を計算する方法は与えられているけれども，ページ間または参照間の統計的依存関係に関する細部の仮定を抜いたままでは $\sigma^2(T)$ を解析的に決定することはできない．無論，実際的状況の下では，正規近似が妥当となるための条件が成立しているならば，必要なのは $s(T)$ と $\sigma^2(T)$ を測定することだけである．式(7.3.19)の変数 $\alpha_i$ がたとえ相互に依存しているのであっても，平均値の線形性によって $s(T)$ を $E[\alpha_i(T)]$ の和で表すことが

許される．作業集合の大きさの分散については次の通りである．

(7.3.28) $$\sigma^2(T)=\lim_{t\to\infty}\mathrm{E}[w^2(t,\ T)]-s^2(T)=\sum_{i\in N}\sum_{j\in N}\mathrm{E}[\alpha_i\alpha_j]-s^2(T)$$

明らかに，共分散(covariance)に関する仮定がなくては $\sigma^2(T)$ を評価することができない．この仮定は，式(7.3.1)の遷移確率 $g_{ij}(t,\ x)$ に関する仮定から導かれる．

正規近似をさらに一層正当化するために，作業集合の大きさの確率過程を，次式で定義される確率変数 $\theta(t,\ T)\ (t\geq 1)$の系列を用いて表現しよう．

(7.3.29) $$w(t,\ T)=w(t-1,\ T)+\theta(t,\ T)$$

作業集合の定義に従えば，$\theta(t,\ T)$ の値は $-1, 0, 1$ のいずれかに限ると仮定することができる．時刻 $t$ における作業集合の大きさは，この系列を用いて次のように表される．

(7.3.30) $$w(t,\ T)=\sum_{k=1}^{t}\theta(k,\ T)$$

古典的な中心極限定理を式(7.3.30)に適用するには，系列 $\{\theta(t,\ T)\}_{t\geq 1}$ が互いに独立な確率変数から成っていることが必要である．明らかに，この系列はそうはなっていない．しかしながら，参照が漸近的に無相関になって行くという仮定は，$\theta(t,\ T)$ と $\theta(t+x,\ T)$ が $x\to\infty$ では無相関になると推論することを許し，このことが中心極限定理の論法の適用を十分に可能にする．与えられた正の整数を $q$ と置くと，式(7.3.30)は(大きな $t$ に対して)次式のように書き表せることが看取できる．

(7.3.31) $$w(t,\ T)=\sum_{r=1}^{q}\sum_{k\geq 0}\theta(kq+r,\ T)$$

$q$ を十分に大きく選ぶことにより，部分系列 $\{\theta(kq+r,\ T)\}_{k\geq 0}$ の引き続く項の間の相関を任意に小さくできる．したがって，式

(7.3.32) $$w_r=\sum_{k\geq 0}\theta(kq+r,\ T)\quad (r=1,\ 2,\ \cdots,\ q)$$

は正規過程によって近似される．このとき，作業集合の大きさは次式のようになる．

(7.3.33) $$w(t,\ T)=w_1+\cdots+w_q$$

この式は，(互いに独立でない)正規過程の和になっているけれども，それ自身

は正規である．その平均は漸近的に $s(T)$ であるが，分散は項 $w_1, \cdots, w_q$ の間の相関に関する仮定なしには明示することができない．

式(7.3.29)から(7.3.33)までの議論は，$w(t, T)$ が近似的に正規であることの形式的証明ではない．漸近的無相関性よりももっと一般性のある仮定を用いた形式的証明は文献[19]に見られる．その細部は込み入っており，本節の目的に対して特に啓発的であるというのでもない．漸近的無相関性の仮定が成立しないときには，$w(t, T)$ の正規近似は粗雑なものになるであろうことに留意するのは大切である．文献[20]に述べられているある種のプログラムは，この漸近的無相関性という性質を持たず，その作業集合の大きさは正規分布に従わない．大きな大域的ループを持つプログラムというものは，この性質を満たさないし，またその作業集合の大きさが正規分布に従うこともない．

## 7.4　LRU ページングと作業集合の関係

作業集合 $W(t, T)$ の内容は，時刻 $t$ におけるプログラムの LRU(least recently used)スタックの先頭からの $w(t, T)$ 個の項を取ることにより決定することができるということを先に(7.3.2 節で)注意しておいた．すなわち，作業集合アルゴリズムをシミュレートするために，LRU アルゴリズムを使うことができる．これは，LRU スタックを次のように修正することにより行うことができる．$j$ は最終参照が最も最近から数えて $i$ 番目に行われたページである† とし，$j$ の最終参照が行われた時刻を $t_j$ と置く．各 $i$ に対して，修正された LRU スタックの $i$ 番目の要素は対 $s_i = (j, t_j)$ であると定義する．$T$ は与えられているとすると，時刻 $t$ における作業集合の大きさは，$s_z = (j, t_j)$ かつ $t_j \in [t - T + 1, t]$ が成り立つような最大の整数 $z$ に他ならない．

LRU ページング・アルゴリズムは，逆に作業集合モデルによってシミュレートすることもできる．$p$ ページの大きさの LRU 記憶の内容は，常に，最も最近に参照された $p$ ページから成り，一方，作業集合 $W(t, T)$ の内容は，いつも，最も最近に参照された $w(t, T)$ 個のページから成るという点にシミュレーションの根拠がある．このことは，$p$ ページの大きさの LRU 記憶に対するページ落ち発生の確率(page-fault probability)を見積るために，平均の大きさが $p$ である作業集合に対するページ不在の確率(missing-page probability)を使用することができるということを示唆している(図 7.4-1 参照)．すなわち，

†訳注　すなわち，$j$ は後向き距離が $i$ 番目に小さいページである．

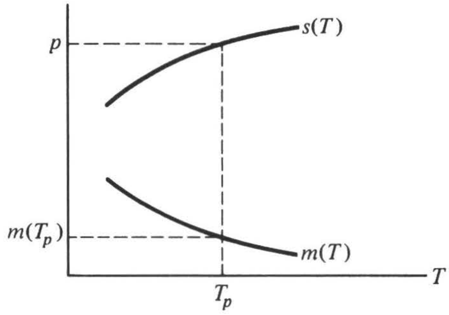



図 7.4-1 LRU ページング率の見積り

$$F_p(\mathrm{LRU}) \cong m(T_p), \quad s(T_p) = p \tag{7.4.1}$$

LRU ページングと作業集合の関係のその他の諸特質に関しては演習問題で研究することにしよう.

下記の二つの例では，ページ落ち発生の確率を指定された範囲内に収めるように制御するためには，これまでの議論をどのように応用し拡張すればよいかを示している.

**例**: $m_1 > m_2$ とし，高い確率で $m_1 \geq m(T) \geq m_2$ となるように $T$ を制御することが望まれていると仮定する．すなわち，$t_{i-1} \leq t < t_i$ および $m_1 \geq m(T_i) \geq m_2$ に対して，作業集合が $W(t, T_i)$ となるように時刻の系列 $t_0t_1\cdots t_i\cdots$ と窓の大きさの系列 $T_1T_2\cdots T_i\cdots$ を見出すことが望まれているとする．この在り方を近似する一つの方法は，以下のように操作することである．窓の大きさの増加分を $T_0$ と置く．$T_1$ は与えられていると仮定し，$t_i = t_{i-1} + T_i$ により各 $t_i$ を定める．$(t_{i-1}, t_i)$におけるページ不在の確率を次式

$$\hat{m}(T_i) = \frac{(t_{i-1},\ t_i)\text{における } W(t,\ T_i)\text{への入来数}}{T_i}$$

を用いて見積り，下記の決定規則に従って $T_{i+1}$ を定める.

もし　$\hat{m}(T_i) > m_1$ ならば，$T_{i+1} = T_i + T_0$
もし　$m_1 \geq \hat{m}(T_i) \geq m_2$ ならば，$T_{i+1} = T_i$
もし　$m_2 > \hat{m}(T_i)$ ならば，$T_{i+1} = T_i - T_0$

**例**: 上述の例の決定規則を LRU ページングで次のように近似することができる．LRU の下でページ落ちが発生する時点の系列を $t_0t_1\cdots t_i\cdots$ と置く．$(t_{i-1}, t_i)$において LRU が記憶装置内に持つページの数を $p_i$ と置くと，記憶領域の大きさの系列 $p_1p_2\cdots p_i\cdots$ が定義されることになる．ページ落ち

発生の確率を

$$\hat{F}_i = \frac{1}{t_i - t_{i-1}}$$

で見積り，下記の決定規則に従って $p_{i+1}$ を定める．

もし　$\hat{F}_i > m_1$ ならば，$p_{i+1} = p_i + 1$(記憶領域にページを追加する)
もし　$m_1 \geq \hat{F}_i \geq m_2$ ならば，$p_{i+1} = p_i$(LRU ページを入れ換える)
もし　$m_2 > \hat{F}_i$ ならば，$p_{i+1} = p_i - 1$(LRU ページを入れ換え，かつ1ページ減らす)

## 7.5　固定分割対可変分割[18]

本節では，多重プログラミング下の記憶領域管理に前節までの結果を応用する問題を取扱う．主に関心の対象となる設計問題は，活動中のプログラムの間で記憶領域を分割する際に用いる方策である．作業集合による記憶領域の管理が最適であるかまたはそれに近いとするならば，ページ落ち発生確率の制御という見地から，記憶領域を要求するプログラムの間での領域分割問題を解決しておかなければならない．この問題は，システム性能に及ぼす影響という点で少なくともページ排除のアルゴリズム選択問題と同じくらい重要であるように思われる．

特に作業集合モデルを通して顕示される局所性の諸概念は，記憶領域分割の二つの基本的方策を比較する際に利用されることになる．一つの方策は**固定分割法**(fixed partitioning strategy)と言われるものであり，(最大)多重度((maximum)degree of multiprogramming)が固定され，記憶領域は活動中のプログラムの間で等大に分割されるという方式である．この方策によれば，活動中のプログラムの各々は厳格にそれ自身の専用区画内に封じ込められることになる．他の一つは**動的分割法**(dynamic partitioning strategy)と呼ばれるものであって，プログラムの各々は，作業集合の大きさに合わせて動的に変動する記憶領域量を割当てられる．実用的な領域分割方策の多くはこれら両極端かその中間に当る方式になっている．動的分割法は記憶領域管理の作業集合原理を実現する一つの形態であるということに注意しなければならない．

固定分割法を支持する第一の論拠は，実現の簡単さとその低オーバヘッド性にある．直観的には，もしも，プログラムのそれぞれを封じ込める区画の大きさが固定されてはいるものの，動作時間の大部分はプログラムの作業集合を含む大きさを十分に持っているとするならば，固定分割方式は満足に作動するは

ずである.

ところが，それに反して区画の大きさが小さ過ぎるならば，飽和していない区画の中の使われていない空間の割当てを為直して，溢れの生じた状態で働く区画に空間を増強することができないので，性能を低下させるような影響が出るであろう．下記で，固定分割の区画の大きさを小さめに取ると，動的分割に較べて性能が著しく悪化するということを示そう．このような場合には，システムの性能と容量の改善に要する費用は動的分割の実現に要する追加費用を却って大きく上回ることにもなろう.

与えられたプログラム $i$ の作業集合の大きさを $w_i(t)$ で表そう．**7.3.4** 節で展開した論述により，$w_i(t)$ は，平均 $s = s(T)$ と分散 $\sigma^2 = \sigma^2(T)$ を持つ正規分布に従う．$\Phi$ で零平均と単位分散を持つ正規確率変数の累積分布を表すと，$w_i(t)$ の累積分布を次のように表現することができる.

$$\Pr[w_i(t) \leq x] = \Phi\left(\frac{x-s}{\sigma}\right) \tag{7.5.1}$$

$n$ 個のプログラムが滞留する $M$ ページの記憶領域を考え，この $n$ 個のプログラムの互いに独立な作業集合の大きさ $w_1(t), \cdots, w_n(t)$ はそれぞれ共通のパラメータ $s$ と $\sigma$ を持つと仮定しよう．(最後のこの仮定は，必然性があって置かれたのではなく，むしろ便宜のために置かれている.) 固定分割の場合，$M = nb$ と置くと，各プログラムにはそれぞれ $b$ ページの空間が割当てられることになり，可変分割の場合，各プログラムにはそれぞれその作業集合の大きさに等しい量の空間が割当てられることになる．各場合に対して尺度を二つ定義する．すなわち，**安全率**(safeness factor)は溢れの条件が存在しない確率を表し，**超過量**(exceedance)は総溢れ量の平均である．所与の尺度に関して二つの場合を比較すると，可変分割は，常に，固定分割と少なくとも同程度に働き，また作業集合の大きさの分散が大になるにつれ目覚ましい差違が現れることが分る.

固定分割の場合を考えよう．$n$ 個の独立なプログラムを **1** 個の結合体と見たときの振舞は，各時刻 $t$ での記憶領域使用状況(storage configuration)

$$\mathbf{Z}(t) = (Z_1(t), \cdots, Z_n(t)),\ \text{ただし,} \tag{7.5.2}$$

$$Z_i(t) = \min\{b, w_i(t)\} \quad (i = 1, \cdots, n) \tag{7.5.3}$$

によって明示される．領域使用状況 $\mathbf{Z}(t)$ は，いくつかのプログラムが($n$ プロセッサによって)同時に動作するシステム，あるいはまた，計時尺度(時計の

針の進ませ方)を適当に変更すれば，($m(<n)$，おそらくは1プロセッサの)プロセッサ共有(processor sharing)(4.4節参照)の直接的なモデルになる．より長期にわたる時間の上では，これは一つのプロセッサが(ラウンドロビン・スケジューリング方策の下でのように)プログラム間で多重使用(交互に使用)されるような多重プログラミング・システムのモデルにもなる．個々の区画に関しては，作業集合原理に対応する記憶領域割当ての手続きが有効であることを上記は暗黙の内に述べている．たとえば，LRU ページング・アルゴリズムが一つのプログラムのどの部分をその分割区画上に搭載せしめるかを決定する．換言すれば，$\mathbf{Z}(t)$ の定義に従って，実行中の各プログラムの作業集合の最大部分がそれぞれの区画内に維持されるといった程度には作業集合に基づく記憶領域管理が行われることになる．

分割区画は，与えられた時点で，対応するプログラムの作業集合が収まるならば，**安全である**(safe)と考えられる．式(7.5.3)によれば，区画 $i$ は，$Z_i(t) = w_i(t)$ が成り立つならば安全である．固定分割システムは，$n$ 個の区画の全てが同時に安全であるときには常に安全であると考えることにする．作業集合の大きさの確率密度として $h(z, T)$ (式(7.3.25)を参照せよ)を用いると，この場合の安全率は次式のように定義される．

$$S_f = \left[\int_{-\infty}^{b} h(z, T)dz\right]^n = \Phi^n\left(\frac{b-s}{\sigma}\right) \tag{7.5.4}$$

ただし，積分は1区画が安全である確率であり，ここでは，区画間の統計的独立性の仮定を援用している．$s$ の関数としての $S_f$ の振舞は**図7.5-1**に示される．この図の齎す情報は次の文に要約されている．固定分割方式は，作業集合の大きさの変動が(平均値に対して)小であり，かつ作業集合の平均の大きさよりも大雑把に言って $2\sigma$ だけ大きい区画の大きさ $b$ が用いられるときには許容できる安全性を与える．これらの条件の下では記憶領域利用率(storage utilization)は高くなるが，しかし，飽和した(安全でなくなった)区画によって惹起されるページング率の増大は目に立つ程でもない．

ページの出入りがもっと直接的に評価できる性能尺度——**平均超過量**(mean exceedance)——は，$n$ 個の作業集合が供給量を超過して必要とする空間の平均量である．次式

$$X_i(t) = \max\{0, w_i(t) - b\} \tag{7.5.5}$$

を定義すれば，固定分割に対する平均超過量は次の式によって与えられる．

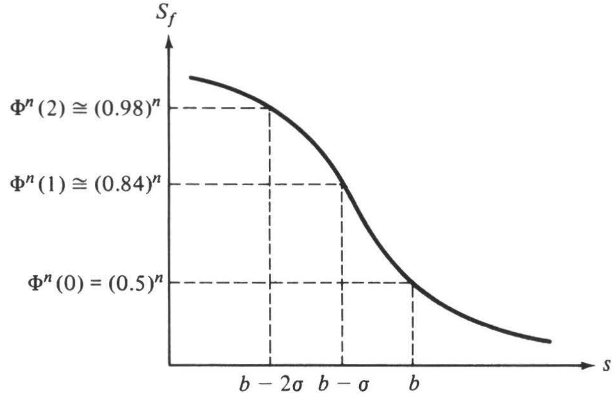



図 7.5-1　$S_f$ の振舞

$$E_f = \mathrm{E}[\sum_{i=1}^{n} X_i(t)] = n\int_b^\infty (z-b)h(z,\ T)dz$$

式(7.3.25)を用いて簡単化すると，次式が得られる．

$$E_f = \frac{n\sigma}{\sqrt{2\pi}}e^{-(s-b)^2/2\sigma^2} + n(s-b)\Phi\left(\frac{s-b}{\sigma}\right) \tag{7.5.6}$$

可変分割に対する安全率 $S_v$ と超過量 $E_v$ は以下のように定義される．それぞれがパラメータ $s$ と $\sigma$ を持つ $n$ 個の正規分布に従う確率変数の和が $M$ を超えない確率がこの場合の安全率 $S_v$ となる．$n$ 個の独立な正規分布に従う確率変数の和はそれ自身平均 $ns$ と分散 $n\sigma^2$ を持つ正規分布に従うから，安全率と超過量は次式のように表現することができる．

$$S_v = \int_{-\infty}^{M} h_n(z,\ T)dz = \Phi\left(\frac{M-ns}{\sqrt{n}\,\sigma}\right) \tag{7.5.7}$$

および

$$E_v = \int_M^\infty (z-M)h_n(z,\ T)dz$$

ここで，$h_n$ はパラメータ $ns$ および $n\sigma^2$ を持つ正規分布である．$E_v$ のこの表現式は次のように簡単化することができる．

$$E_v = \sigma\sqrt{\frac{n}{2\pi}}\,e^{-(ns-M)^2/2n\sigma^2} + (ns-M)\Phi\left(\frac{ns-M}{\sqrt{n\sigma^2}}\right) \tag{7.5.8}$$

$S_f$ と $S_v$ の比較は**図 7.5-2** に示されている．そこには，平均 $s$ に対する $\sigma$ の相対的な値が大であるときに可変分割によって著しい改善が見られることが特徴的に示されている．次式

$$\sigma \leq \frac{b-s}{2} \tag{7.5.9}$$

が成り立つとき，すなわち，条件 $b \geq s+2\sigma$ が成立するときは，$S_f$ と $S_v$ は常に同程度の値を示す．($\sigma \to 0$ とするならば，この二つのシステムは漸近的に等価になるということは容易に確かめられる．)性能比 $S_v/S_f$ は，$\sigma$ が値$(b-s)/2$ を超えて次第に大きくなるにつれて急速に増大する．

**図 7.5-3** は，ある特定のパラメータ値を選択したとき，与えられた水準の超過量に対応して，記憶領域の必要量が二つの方式によってどの程度に異なるかを示している．$M$ の値が小さいときは，固定分割方式における全ての区画も動的分割方式における全記憶領域も共に飽和してしまうから，$E_f$ と $E_v$ は漸近的に同じ値になる．上述からも知られるように，$b=s+2\sigma$ が成り立つときが固定分割に対しては最も興味がある．$b=s+2\sigma$ かつ $M=nb$ に対する $E_f$ 曲線上の点を截る水平な直線が図 7.5-3 上に示してある．たとえ $S_f$ と $S_v$ とがほとんど同じ値であっても，この場合に対する超過量水準を保つために，可変分割方式では前記の $M$ の約70% の領域しか必要としないということが明らかになる．このことは，作業集合の大きさの分散が比較的に大になると，無視できない程の領域の節減が動的分割によって可能になるということを直接的

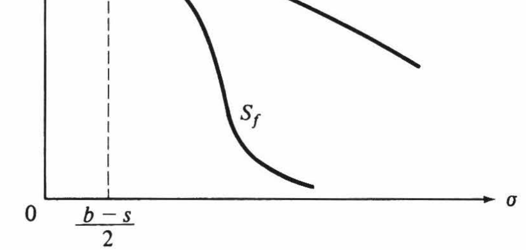



**図 7.5-2**　安 全 率

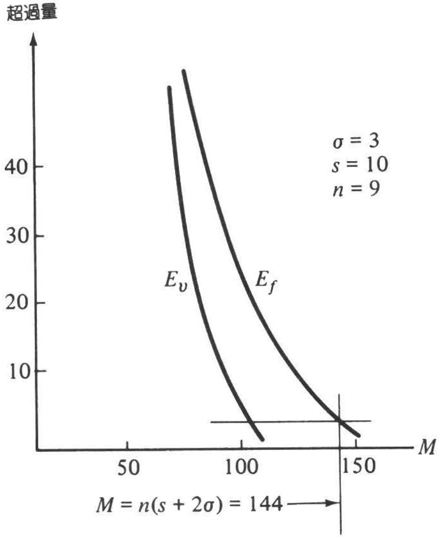



図7.5-3 超 過 量

に示している.

動的方式のもう一つの重要な利点は，作業集合の大きさの分布に変動があるとき，すなわち，パラメータ $s$ および $\sigma$ が非定常的 $[s = s(t, T)$ かつ $\sigma = \sigma(t, T)]$ であるときに対するこの方式の適応能力にある．固定分割方式では，下記の条件が成立していない期間にはその性能が著しく低下してしまうことがあり得る.

$$b \geq s(t, T) + 2\,\sigma(t, T)$$

動的方式では，その性能がそれ程まで低下するようなことはない.

ここまでの議論は，多重度は一定値に固定し，その代り，ページング率が変動することを許すという方式を取扱ってきた．逆を考えることも可能である．すなわち，一部の(可能な限り多数の)プログラムの作業集合の総和(和集合)のみを記憶領域内に保持することによってページング率を一定に保ち，その結果として多重度が変動することは許すという方式である．このような方式では統計量 $E[n(t)]$ に関心が持たれる．これを評価するため，次の2値確率変数を定義しよう.

$$\gamma_k(t) = \begin{cases} 1 & (w_1(t) + \cdots + w_k(t) \leq M \text{ のとき}) \\ 0 & (\text{そうでないとき}) \end{cases} \tag{7.5.10}$$

そうすると，

$$n(t)=\sum_{k=1}^{\infty}\gamma_k(t) \tag{7.5.11}$$

したがって，次の結果を得る†.

$$\mathrm{E}[n(t)]=\sum_{k=1}^{\infty}\Phi\left(\frac{M-ks}{\sqrt{k}\,\sigma}\right) \tag{7.5.12}$$

実用のためにこの方式を実現するに際しては，境界 $M$ における不安定効果に無関心であってはならない．たとえば，溢れが生じる正にそのときに，追加プログラムが活性化される(活動を始める)場合を考えよう．そうすると，作業集合の大きさの和が値 $M$ の辺りで統計的に生じるゆらぎが原因となって，活性化されたり不活性化されたりするプログラムの激しい出入りを招来することになる．このような状況を回避するためには，空いている空間が作業集合を包含するのに必要な量 $H$ を超過しているときにしか，新たなプログラムを導入してはならないとするとよい．$H$ は割当て手続きのパラメータになる．記憶領域の大きさ $M$ と値 $M-H$ に対する式(7.5.12)の値はそれぞれこの場合の $\mathrm{E}[n(t)]$ の上界と下界のよい見積りとして使用するができる．

動的分割法の実用的な実現においては，多重度が固定されていても，また固定されていなくても，どちらの場合であっても，上限突破(記憶領域溢れ)の発生率の平均値がもう一つの重要な性能の尺度になる．というのは，上限突破は，システム経費——作業集合の大きさを減少させるか，または多重度を調整

†**訳注**　式(7.5.12)を得るためには，多少の計算が必要である．式(7.5.1)から

$$F(M)=\Pr[w_1(t)\le M]=\Phi\left(\frac{M-s}{\sigma}\right) \tag{A}$$

$n(t)$ の定義から

$$\Pr[n(t)\ge k]=\Pr[w_1(t)+\cdots+w_k(t)\le M]=F_k(M) \tag{B}$$

ただし $F_k(M)$ は

$$F_k(M)=\int_0^M F_{k-1}(M-x)\,dF(x)=\Phi\left(\frac{M-ks}{\sqrt{k}\,\sigma}\right),\quad F_1(M)=F(M) \tag{C}$$

で定義される．式(B)から，多重度の確率塊関数

$$\begin{cases}P_k(M)=\Pr[n(t)=k]=F_k(M)-F_{k+1}(M)\\ P_0(M)=1-F(M)\end{cases} \tag{D}$$

が得られる．したがって，平均多重度は次のようになる．

$$\begin{aligned}\mathrm{E}[n(t)]&=\sum_{k=1}^{\infty}kP_k(M)=\sum_{k=1}^{\infty}k[F_k(M)-F_{k+1}(M)]\\&=\sum_{k=1}^{\infty}F_k(M)=\sum_{k=1}^{\infty}\Phi\left(\frac{M-ks}{\sqrt{k}\,\sigma}\right)\end{aligned}$$

するために要する経費——が発生する時点を表しているからである．なお，上限突破問題の解析は演習問題 5.9 で扱われている．

## 演 習 問 題

**7.1** この問題では，(本文中で行ったような)空間平均(ensemble average)の代りに時間平均(time average)を用いて作業集合の大きさの平均 $s(T)$ とページ不在確率 $m(T)$ を定義することの意味を考える．まず，次式を定義する．

$$s_k(T) = \frac{1}{k}\sum_{t=1}^{k} w(t,\ T),\quad m_k(T) = \frac{1}{k}\sum_{t=0}^{k-1}\Delta(t,\ T)$$

ただし，$\Delta(t,\ T)$ は，$r_{t+1} \notin W(t,\ T)$ であるときかつそのときに限り値 1 を取る 2 値変数である．極限

$$s(T) = \lim_{k\to\infty} s_k(T), \qquad m(T) = \lim_{k\to\infty} m_k(T)$$

を取ると，関係式

$$s(T+1) - s(T) = m(T)$$
$$s(T) = \sum_{x=0}^{T-1} m(x)$$

が成立することを証明せよ．次に，プログラムは $n$ 個の再帰的ページから成ると仮定して，次式が成り立つことを示せ．

$$\lim_{T\to\infty} s(T) = n$$
$$\lim_{T\to\infty} m(T) = 0$$

**7.2** $\Delta(t,\ T)$ を演習問題 7.1 におけるそれと同じに定義する．まず，$\Delta(t,\ T+1) \le \Delta(t,\ T)$ を証明し，次に，この式を用いて，$m(T+1) \le m(T)$ が成り立つという結果が得られることを示せ．

**7.3** 演習問題 7.1 の定義では，$t=1$ から $t=k$ までの時間間隔が用いられた．その代りにここでは，時間間隔 $t=t_0+1$ から $t=t_0+k$ までを用いよう．ただし，$t_0 \ge 0$ は任意であるとする．このときは，

$$s_k(T) = \frac{1}{k}\sum_{t=t_0+1}^{t_0+k} w(t,\ T), \qquad m_k(T) = \frac{1}{k}\sum_{t=t_0}^{t_0+k-1}\Delta(t,\ T)$$

である．まず，次式が成り立つことを示せ．

$$s_k(T+1) = s_k(T) + m_k(T) - \frac{w(t_0+k,\ T) - w(t_0,\ T)}{k}$$

$$\frac{s_k(T-1) + s_k(T+1)}{2} \leq s_k(T) + \frac{1}{2k}(w(t_0+k,\ T-1)$$
$$- w(t_0,\ T-1) - w(t_0+k,\ T) + w(t_0,\ T))$$

次に，$s_k(T)$は，$k \to \infty$ でない限り，下に凹であるとは限らないことを示せ．たとえば，参照列を

$$\omega = 123452133234545$$

とし，$t_0 = 5$ および $k = 10$ の場合を考えよう．しかし，$t_0 = 0$ のときは，$w(t_0,\ T) = 0$ であるので，$s_k(T)$ は下に凹となることを示せ．

**7.4** この問題では，LRU アルゴリズムと作業集合モデルの関係について調べよう．可変長窓 $T(t,\ p)$ は $w(t,\ T) = p$ が成り立つ最小の $T$ であると定義する．ここで，$p$ は整数である．$W(t,\ T(t,\ p))$ は，正に時刻 $t$ における LRU スタックの先頭からの上位 $p$ 個の要素に他ならないことをまず看取せよ．$g_p(x) = \lim_{t\to\infty}\Pr[T(t,\ p) = x]$ を定義し，時刻の系列 $\{t_k(x)\}$ は $T(t,\ p) = x$ となるような $t$ の系列であると定義する．また，$m(T)$ を全ての $t$ の上で定義したのと同様に，$\{t_k(x)\}$ の上のページ不在確率 $m_p(T)$ を定義し，さらに，$T$ の平均値を $T_p = \sum_{x>0} xg_p(x)$ と定義する．

**a**. $p$ ページから成る LRU 記憶領域に対するページ落ち確率は

$$F_p(\mathrm{LRU}) = \sum_{x>0} g_p(x) m_p(x)$$

であることを示せ．

**b**. 窓の大きさ $T$ に対する総合ページ不在確率(overall missing-page probability)は

$$m(T) = \sum_{i \in N} g_i(T) m_i(T)$$

であることを示せ．

**c**. a 項と b 項を用い，($n$ ページのプログラムに対し)次式が成り立つことを示せ．

$$\sum_{x>0} m(x) = \sum_{i \in N} F_i(\mathrm{LRU}) = n$$

**d.** $m(T_p) \leq F_p(\mathrm{LRU})$ が成り立つような $p$ の値が少なくとも1個は存在すること，すなわち，作業集合による LRU のシミュレーションはより小さなページ不在確率を結果として生ぜしめることを示せ．

**e.** $a_1, \cdots, a_n$ は LRU スタック・モデルの距離分布を表すとしよう．式

$$m(T_i) - m(T_{i+1}) \leq a_{i+1} \quad (i \geq p)$$

は，$m(T_p) \leq F_p(\mathrm{LRU})$ の成立を保証するための十分条件となることを示せ．また，$a_{i+1} = m(T_i) - m(T_{i+1})$ は一般には真でないことを示せ．

**7.5** 非再帰的ページ($\beta_i = 0$ であるページ)は作業集合の大きさの長時間走行平均 $s(T)$ に何の影響も及ぼさないことは既に言及した．これを示すため，$r_1 r_2 \cdots r_k$ の上でのページ $i$ に対する参照間隔の密度関数 $f_{ik}(x)$ をまず明確に定義し，そこでの参照がページ $i$ に対してなされる割合を $\beta_{ik}$ で表すとせよ．非再帰ページ $i$ に対して

$$\lim_{k \to \infty} \beta_{ik} f_{ik}(x) = 0$$

が成り立つことを示し，次に，これを用いて非再帰的ページは

$$s(T) = \sum_{i \in N} \sum_{x=0}^{T-1} \beta_i (1 - F_i(x))$$

に寄与しないという結論を求めよ．$\bar{x}_{ik}$ は $r_1 r_2 \cdots r_k$ 内のページ $i$ に対する平均参照間隔を表すとしよう．ページ $i$ は非再帰的であると仮定し，次の式を証明せよ．

$$\lim_{k \to \infty} \beta_{ik} \bar{x}_{ik} = 0$$

(証明を完全なものにするには定常性を援用する必要がある．)

**7.6** 第6章の演習問題で，記憶容量を固定したままページの大きさを半分にすることはページ落ち率を2倍以上に増大させ得るということを示した．作業集合モデルにおいても同じ型の振舞を観察することができる．$s_a(T)$ は，ページの大きさが $2^a$ 語であるときの作業集合の大きさの期待値を表し，$m_a(T)$ はそのときのページ不在確率であるとする．

**a.** $2s_{a+1}(T) \geq s_a(T) \geq s_{a+1}(T)$，すなわち，ページの大きさを半分にする

と，ページを単位として計られる作業集合の大きさ(ページ数)は増大するが，2倍以上にはならないことを示せ．また，$2^a \cdot s_a(T) \leq 2^{a+1} \cdot s_{a+1}(T)$，すなわち，ページの大きさを半分にしても，作業集合内の総語数は増大しないことを示せ．

**b.** $\omega_1\omega_2\cdots\omega_n\omega_1$ は，$\omega_i$ の終りの $n-1$ 個の数字が $\omega_{i+1}$ の始めの $n-1$ 個の数字に一致するような $12\cdots n$ の巡回置換(cyclic permutation)の系列であるとする．列 $\omega'_i$ と $\omega''_i$ は，それぞれページ集合 $\{1', \cdots, n'\}$ と $\{1'', \cdots, n''\}$ の上の列であるという点以外は $\omega_i$ と同じである．ここで，$j'$ と $j''$ はページ $j$ を構成する相棒(buddy)である．$n$ は偶数であると仮定して，参照列

$$\omega(a) = \alpha_k = (\omega'_1\omega''_2\omega'_3\omega''_4\cdots\omega''_n)^k$$
$$\omega(a+1) = \beta_k = (\omega_1\omega_2\cdots\omega_n)^k$$

を定義する．$k \to \infty$ とすると，次式が成り立つことを示せ．

$$\frac{m_a(T)}{m_{a+1}(T)} = n \quad (T = n-1)$$

また，$k \to \infty$ とすると，次式が成り立つことも示せ．

$$\frac{m_a(2\,T)}{m_{a+1}(2\,T)} = n \quad (T = n-1)$$

(すなわち，ページの大きさを半分にした後に窓の大きさを2倍にしても変則性を取り除かない．)

**7.7** ページ不在率に対するページの大きさの影響をさらに調べよう．$m_i(T) = \beta_i(1 - F_i(T))$ はページ $i$ の不在率であるとする．したがって，$m(T) = m_1(T) + \cdots + m_n(T)$ となる．ページの大きさ $2^a$ に対する $m_i(T)$ を $m_i{}^a(T)$ と書くことにする．

**a.** $m_i{}^a(T) \leq m_i{}^{a+1}(T)$ は，直観的に望ましい結果 $m^a(T) \leq 2\,m^{a+1}(T)$ が成り立つための十分条件となることを示せ．$i'$ と $i''$ はページ $i$ を構成する相棒を表すものとすると，この条件は次式と等価である．

$$\beta_j(1 - F_j(T)) \leq \beta_i(1 - F_i(T)) \quad (j = i' \text{ または } i'')$$

**b.** 前出の問題で示されたように，a 項の条件は成立するとは限らない．比較的に振舞が良性の独立参照モデルに対してすらこれが成立するとは限らないということは，次のようにして立証することができる．$m_i(T)$

$=\beta_i(1-\beta_i)^T$ であることに注意して，$m_i(T)$ は，$\beta_i=1/(T+1)$ に対して最大となり，それより大きい $\beta_i$ に対しては減少することを示せ．ページを半分の大きさに切り離すことの効果は，その結果得られる一組の相棒 $i'$ と $i''$ に対する $\beta_i$ の値を減少させることであるから，もし $\beta_{i'}$ と $\beta_{i''}$ が両方とも $1/(T+1)$ より大であるならば，$j=i'$ または $i''$ に対して $m_j{}^a(T)>m_i{}^{a+1}(T)$ となる．これを用いて，$m^a(T)$ は $2\,m^{a+1}(T)$ より大となり得るという結論を導け．

**7.8** ページ参照確率分布 $\beta_1,\cdots,\beta_n$ を持つ独立参照モデルによってモデル化されるプログラムを考える．集合 $L_{m-1}$ は $\beta_i$ の値の大きい順番に選んだ $(m-1)$個のページの集合であると定義する．最適アルゴリズム $A_0$ の研究(第6章)の際に，平衡状態においては，$L_{m-1}$ に属するページは常に記憶領域内にあって，1回の参照当りのページ落ち確率が

$$C(m)=\sum\beta_i-\frac{\sum\beta_i{}^2}{\sum\beta_i}$$

となることを示した．ただし，総和はどれも $N-L_{m-1}$ のページについて取られる．さて，(定常状態における)時刻 $t$ に，ページのそれぞれに結び付けられている確率の置換が生じ，分布 $\{\beta'_i\}=\{\beta_i\}$ が新たに再定義される(すなわち，新たな局所性集合が導入される)と仮定しよう．時刻 $t+T$ におけるページ落ち確率は $C(m)+g(T)$ と表すことができる．ただし，$C(m)$ は上記と同じ定常状態確率であり，$g(T)$ は $L'_{m-1}$ のページが主記憶領域内へ呼び戻されて溜って行く過程に対応して減衰的に経過する過渡状態確率である．

過渡状態確率 $g(T)$ は作業集合の議論を用いてかなり簡単に計算できる．集合 $M=L'_{m-1}-L_{m-1}$ は，参照確率の置換の後，やがて定常状態に達したとき記憶領域内に配置されているべきページの内，置換直後には領域内に置かれていないものの集合であると定義される．$g(t)$ は集合 $M\cap W(t+T,T)$ に対するページ不在率であることを示し，次式が成り立つという結果を導け．

$$g(T)=\sum_{i\in M\cap W(t+T,T)}\beta'_i(1-\beta'_i)^T$$

$\{\beta'_i\}$ が新たに時刻 $t$ から有効になる任意の分布であってよいように上述の議論の拡張を試みよ．

**7.9** $(l, \alpha)$局所性モデルは以下のように定義される[17]．ここでは，$N$の丁度$l$ページを含む部分集合$L$を**局所性集合**という．$L$の**外部**ページ$x(x \notin L)$と**内部**ページ$y(y \in L)$に対し，$L' = L + x - y$が成り立つとき，局所性集合$L$および$L'$は必要時ページングにおける隣接集合であるという．参照列$r_1r_2\cdots r_{t-1}$および局所性集合の系列$L_0L_1\cdots L_{t-1}$が生成されているとする．このとき，確率$1-\alpha$で$x$が$L_{t-1}$の要素の中からランダムに選び出されるか，または，確率$\alpha$で対$(x, y)$が$(N - L_{t-1}) \times L_{t-1}$の要素の中からランダムに選び出されると仮定される．いずれにしても$r_t = x$ではあるが，前者の場合には$L_t = L_{t-1}$となり，後者の場合には$L_t = L_{t-1} + x - y$となる．与えられた局所性集合の持続時間(holding time)の分布は幾何分布

$$h(v) = (1-\alpha)^{v-1}\alpha \quad (v \geq 1)$$

に従うから，局所性集合間の遷移確率はいつの時点でも値$\alpha$を取り，$1/\alpha$が平均持続時間となることに注目しよう．遷移は必要時ページングにおける隣接集合の間にだけ生じるということにも注意しよう．局所性集合のページに参照が偏るという要件に適合させて，次式，

$$\frac{1-\alpha}{l} \geq \frac{\alpha}{n-l}$$

すなわち，内部ページを参照する確率は外部ページを参照する確率よりも小ではないということを仮定する．

記憶領域の大きさを$m \geq l$とし，$m$ページの記憶状態を$S_t$で表す．局所性モデルに従って参照が行われると仮定して，必要時ページングの下で$r_{t+1}\cdots r_{t+k}$を処理するのに要する最小の費用を$C_k(S, L, t)$(ただし，$S_t = S$, $L_t = L$)で表す．第6章の技法に倣って，次式が成り立つことを示せ．ただし，$\bar{S} = N - S$, $\bar{L} = N - L$とする．

$$C_0(S,\ L,\ t) = 0$$

$$C_{k+1}(S,\ L,\ t) = \begin{cases} \sum_{x \in S \cap L} \frac{1-\alpha}{l} C_k(S,\ L,\ t+1) \\ + \sum_{x \in \bar{S} \cap L} \frac{1-\alpha}{l} (1 + \min_{z \in S} C_k(S+x-z,\ L,\ t+1)) \\ + \sum_{x \in S \cap \bar{L}} \frac{\alpha}{n-l} \sum_{y \in L} \frac{1}{l} (C_k(S,\ L+x-y,\ t+1)) \\ + \sum_{x \in \bar{S} \cap \bar{L}} \frac{\alpha}{n-l} \sum_{y \in L} \frac{1}{l} (1 + \min_{z \in S} C_k(S+x-z,\ L+ \\ \qquad x-y,\ t+1)) \end{cases}$$

次の各項をも証明せよ．

**a.** もし全ての時点で方策 $S_t \supseteq L_t$ が用いられるならば，$C_k(S,\ L,\ t) = k\alpha[(n-m)/(n-l)]\ (S \supseteq L)$ である．

**b.** 任意の $S$ に対して，方策‘非局所性ページを排除せよ’は最適である．

**c.** $T = 1/\alpha$ とすると，作業集合 $W(t,\ T)$ は $L_t$ の良好な見積りである．

**7.10** 演習問題7.9のモデルに対する拡張を考えよう．a)$L$ の内部ページ $x$ を参照する確率は $\beta(L,\ x,\ t)$ である．b)遷移が生じてページ $x$ が入来するという条件の下で $L$ から内部ページ $y$ が排除される確率は $\gamma(L,\ x,\ y,\ t)$ である．c)$\alpha$ は $t$ の関数 $\alpha(t)$ で，かつ $\alpha(t)$ は $t \neq t'$ のときは常に $\alpha(t')$ に独立である．なお，以上の拡張においても局所性集合に入来するページ $x$ は $n-l$ 個の外部ページからランダムに選ばれるということに注意しよう．条件

$$(1-\alpha(t))\beta(L,\ x,\ t) \geq \frac{\alpha(t)}{n-l}$$

は方策‘非局所性ページを排除せよ’が最適であるための必要条件であること，また，もし $S \supseteq L$ が成立するならば，この方策に対しては

$$C_k(S,\ L,\ t) = \frac{n-m}{n-l} \sum_{i=1}^{k-1} \alpha(t+i)$$

であることを証明せよ．

**7.11** 演習問題7.9の$(l,\ \alpha)$局所性モデルにおいて，$L_0$ を初期局所性集合とし，また，$M_0 = N - L_0$ とする．参照列を $r_1 r_2 \cdots r_t \cdots$ として，$a_t$ は時刻 $t$ までに参照された $M_0$ の要素の個数であると定義しよう．$a_0 = 0$ で

あること，また，時刻 $t$ において局所性集合間遷移

$$L_t = L_{t-1} + x - y$$

が生じるときに限り $a_t = a_{t-1} + 1$ であることに注意しよう．ただし，$x$ は $M_0 - W(t-1, t-1)$ の要素であり，$W$ は作業集合であるとする．このとき次式

$$\text{(P7.11.1)} \qquad \mathrm{E}[a_t] = (n-l)\left(1 - \left(1 - \frac{\alpha}{n-l}\right)^t\right)$$

が成り立つことを示せ．独立参照モデルにおいては，$k$ ページから成るプログラムに対する作業集合の大きさの平均値は次式

$$\text{(P7.11.2)} \qquad s(T) = k - \sum_{i=1}^{k} (1-\beta_i)^T$$

で表されるということを想起しよう．$k = n - l$, $T = t$, $\beta_i = \alpha/(n-l)$ のときは，式(P7.11.1)と式(P7.11.2)が等しくなることをまず調べ，$a_t$ は実際適当に定義された作業集合の大きさになることを証明せよ．

**7.12** 平均の作業集合の大きさ $s(T)$ は上界 $\min\{n, T\}$ で上から押えられている．参照列が $12\cdots n\,12\cdots n\cdots$ であるときには，実際にこの上界に到達する．しかしながら，独立参照モデルではこの上界に到達することができない．

独立参照モデルにおいてページ参照確率は $\beta_1, \cdots, \beta_n$ であると仮定しよう．まず，

$$s(T) = n - \sum_{i=1}^{n} (1-\beta_i)^T$$

であることを示し，次に，全ての $i$ について $\beta_i = 1/n$ であるときに $s(T)$ は最大となること，すなわち，このモデルに対して次式

$$s(T) \leq n\left(1 - \left(\frac{n-1}{n}\right)^T\right)$$

が成立することを証明せよ．

**7.13** 作業集合の大きさに関する次のモデルを考えよう．解析の便宜上，連続的な時間で過程が進行すると仮定する．作業集合にページが到着(入来)する過程は，到着率 $\lambda$ の指数分布に従い，また，各ページが作業集合内

に滞留する時間は，パラメータ $\mu$ の指数分布に従うと仮定する．作業集合内に $k$ ページが存在する平衡状態確率を $\pi_k$ で表すと，$\pi_k$ は，平均値が

$$s(T) = \rho \qquad \text{(P 7.13.1)}$$

であるポアソン分布

$$\pi_k = \frac{\rho^k}{k!}e^{-\rho} \quad \left(\rho = \frac{\lambda}{\mu}\right)$$

に他ならないことを示せ．同一ページへの引き続く参照の間の時間間隔は互いに独立であって，分布関数 $F(x)$ を持つこと，また，作業集合の窓の大きさが $T$ であると仮定して，ページが作業集合内に滞留する時間の平均が

$$\frac{1}{\mu} = T + \bar{x}\left(\frac{F(T)}{1-F(T)}\right) \qquad \text{(P 7.13.2)}$$

となることを示せ．7.3 節の解析から，ページ不在率(missing-page rate)は

$$\lambda = \frac{d}{dT}s(T) = 1 - F(T) \qquad \text{(P 7.13.3)}$$

となることと $\bar{x} = n$ となることが期待される．しかし，このモデルの限界は次の通りである．1) $\pi_k$ は $k > n$ に対しては小でなければならない．このことは，$\rho$ が $n$ にあまりに近い値であることはできないということを意味する．2) ページの滞留時間は少なくとも $T$ でなければならない．このことは，$T$ が $1/\mu$ に比較して小でなければならないことを意味する．3) ある種の参照時間間隔分布 $F(x)$ のみが式(P 7.13.1—P 7.13.3)と整合する．これらの制限をもっと完全に検討し，分布 $F(x)$ を特徴付けよ．

## 参考文献


1. BELADY, L. A., and C. J. KUEHNER., "Dynamic space sharing in computer systems." *Comm. ACM 12*, 5 (May 1969), 282–288.
2. BRAWN, B., and F. GUSTAVSON, "Program behavior in a paging environment." *AFIPS Proc. Conf.* (1968 FJCC), 1019–1032.
3. DENNING, P. J., "Resource allocation in multiprocess computer systems." M.I.T. Project MAC report MAC-TR-50, M.I.T., Cambridge, Mass. (May 1968).

4. Denning, P. J., "Thrashing: Its causes and prevention." *AFIPS Conf. Proc.* (1968 FJCC), 915–922.

5. Denning, P. J., "Virtual memory." *Computing Surveys 2*, 3 (Sept. 1970), 153–189.

6. Doherty, W., "Scheduling TSS/360 for responsiveness." *AFIPS Conf. Proc.* (1970 FJCC), 97–112.

7. Sayre, D., "Is automatic folding of programs efficient enough to displace manual?" *Comm. ACM 12*, 12 (Dec. 1969), 656–660.

8. Coffman, E. G., Jr., and L. C. Varian., "Further experimental data on the behavior of programs in a paging environment." *Comm. ACM 11*, 7 (July 1968), 471–474.

9. Randell, B., and C. J. Kuehner., "Demand paging in perspective." *AFIPS Conf. Proc.* (1968 FJCC), 1011–1018.

10. Hatfield, D. J., and J. Gerald., "Program restructuring for virtual memory." *IBM Sys. J. 10*, 3 (1971), 168–192.

11. Belady, L. A., "A study of replacement algorithms for virtual storage computers." *IBM Sys. J. 5*, 2 (1966), 78–101.

12. Liptay, J. S., "The cache." *IBM Sys. J. 7*, 1 (1968), 15–21.

13. Oden, P. H., and G. S. Shedler, "A model of memory contention in a paging machine." IBM T. J. Watson Research Center Report RG3053, Yorktown Heights, N.Y. (Sept. 1970).

14. Oppenheimer, G., and N. Weizer. "Resource management for a medium scale time sharing operating system." *Comm. ACM 11*, 5 (May 1968), 313–322.

15. Denning, P. J., "The working set model for program behavior." *Comm. ACM 11*, 5 (May 1968), 323–333.

16. Denning, P. J., and S. C. Schwartz., "Properties of the working set model." *Comm. ACM 15*, 3 (Mar. 1972), 191–198.

17. Denning, P. J., J. E. Savage, and J. R. Spirn., "Models for locality in program behavior." TR-107, Dept. of Electrical Engineering, Princeton Univ., Princeton, N.J. (Apr. 1972).

18. Coffman, E. G., Jr., and T. A. Ryan., "A study of storage partitioning using a mathematical model of locality." *Comm. ACM 15*, 3 (Mar. 1972), 185–190.

19. Rosenblatt, M., "A central limit theorem and a strong mixing condition." *Proc. Natl Acad. Sci. 42* (1956), 43–47.

20. Rodriguez-Rosell, J., "Experimental data on how program behavior affects the choice of scheduler parameters." *Proc. 3rd ACM Symp. on Op. Sys. Princ.* (Oct. 1971).

# 変換，中心極限定理，およびマルコフ連鎖　補遺A

## A.1 母関数

級数

$$A(z)=\sum_{i=0}^{\infty}a_i z^i \tag{A.1.1}$$

によって数列 $\{a_i\}$ の**母関数**(generating function)† が定義される．もし $\{a_i\}$ が確率塊関数(pmf)，すなわち，$a_i \geq 0$，$i \geq 0$，かつ

$$\sum_{i=0}^{\infty}a_i = 1 \tag{A.1.2}$$

を表すならば，$A(z)$ はもっと明確に確率母関数であるという．ここでは，確率母関数だけを扱うので，短く単に母関数といえばこの意味に用いられると了解することにする．明らかに，条件(A.1.2)の下で級数(A.1.1)は少なくとも $0 \leq z \leq 1$ に対しては収束する．ところで，変数 $z$ に特別の意味がある訳ではない．つまり，それはある確率塊関数を表現する関数の単なる独立変数に過ぎない．

さて，母関数は以下の性質を持つ．

$$\lim_{z\to 0}A(z)=a_0 \tag{A.1.3}$$

$$\lim_{z\to 1}A(z)=1 \tag{A.1.4}$$

$X$ は，$\Pr[X=i]=a_i$，かつ $i=1,2$ に対して $\mathrm{E}(X^i)$ が有限であるような

---

† $z$–**変換**($z$–transform)という用語も用いる．

非負の整数値を取る確率変数であるものとする．式(A.1.1)を微分することにより，次式が得られる．

$$A'(z) = \sum_{i=1}^{\infty} i a_i z^{i-1} \tag{A.1.5}$$

この式から，

$$\mathrm{E}(X) = \sum_{i=1}^{\infty} i a_i = \lim_{z \to 1} A'(z) \tag{A.1.6}$$

再び微分すると，次式が得られる．

$$A''(z) = \sum_{i=1}^{\infty} i(i-1) a_i z^{i-2} \tag{A.1.7}$$

この式から

$$\lim_{z \to 1} A''(z) = \mathrm{E}(X^2) - \mathrm{E}(X) \tag{A.1.8}$$

したがって，

$$\lim_{z \to 1} A''(z) = \mathrm{Var}(X) + \mathrm{E}(X)[\mathrm{E}(X) - 1] \tag{A.1.9}$$

母関数の主な利点の一つは，しばしば，$X$ のモーメントが式(A.1.6)，(A.1.8)，および(A.1.9)に示されるように母関数から極めて容易に計算し得るという点にある．その例が**表 A.1-1** に纏められている．高次のモーメントは，それが存在するならば，さらに微分することによって求められることを注意しておく．特に，次式のいわゆる**二項モーメント**(binomial moment)の公式が得られている．

$$\lim_{z \to 1} \frac{d^k A(z)}{dz^k} = \mathrm{E}[X(X-1)(X-2)\cdots(X-k+1)]$$

次に，二つの数列 $\{a_i\}$ と $\{b_i\}$ の**畳込み**(convolution)は，次式によって与えられる数列 $\{c_i\}$ であると定義される．

$$c_i = \sum_{j=0}^{i} a_j b_{i-j} \tag{A.1.10}$$

$X$ と $Y$ は，それぞれ pmf $a_i = \Pr[X = i]$ と $b_i = \Pr[Y = i]$ を持つ二つの互いに独立な非負の整数値を取る確率変数であるとすると，$\{c_i\}$ は，和 $X + Y (c_i = \Pr[X + Y = i])$ の pmf となることが分る．$\{c_i\}$ の母関数 $C(z)$ に関する次の重要な結果は容易に確かめることができよう．

$$C(z) = A(z)B(z) \tag{A.1.11}$$

ここで，$A(z)$ と $B(z)$ は，それぞれ $\{a_i\}$ と $\{b_i\}$ の母関数である．モーメントの計算あるいは互いに独立な確率変数の和の分布の形を求めるときには，式(A.1.10)を扱うよりも式(A.1.11)を取扱う方が大抵は楽である．式(A.1.11)

**表 A.1-1　本書によく現れる確率分布**

(a) 離散分布

| 名　称 | 確率塊関数 | 母関数 | 平均および分散 |
|---|---|---|---|
| 二　項<br>$n=1,2,\ldots,$<br>$0\leq p\leq 1$ | $\binom{n}{j}p^j(1-p)^{n-j},$<br>$j=0,1,\ldots,n$ | $[(1-p)+pz]^n$ | $np$ および $np(1-p)$ |
| ポアソン<br>$\lambda>0$ | $\frac{\lambda^n}{n!}e^{-\lambda}, n=0,1,\ldots$ | $e^{-\lambda(1-z)}$ | $\lambda$ および $\lambda$ |
| 幾　何<br>$0\leq p\leq 1$ | $p(1-p)^n, n=0,1,\ldots$<br>または<br>$p(1-p)^{n-1}, n=1,2,\ldots$ | $p/[1-(1-p)z]$<br><br>$pz/[1-(1-p)z]$ | $(1-p)/p$ および $(1-p)^2$<br><br>$1/p$ および $(1-p)p^2$ |

(b) 連続分布

| 名　称 | 確率密度 | ラプラス変換 | 平均および分散 |
|---|---|---|---|
| (a,b)で一様<br>ただし $0<a<b$ | $\frac{1}{b-a}, a<x<b$<br>$0,$　それ以外 | $\frac{e^{-sa}-e^{-sb}}{s(b-a)}$ | $\frac{a+b}{2}$ および $\frac{(b-a)^2}{12}$ |
| 正　規 | $\frac{1}{\sqrt{2\pi\sigma^2}}e^{-(x-m)^2/2\sigma^2}$<br>$-\infty<x<+\infty$ | † | $m$ および $\sigma^2$ |
| 指　数†† | $\lambda e^{-\lambda x}, x>0$ | $\frac{1}{1+s/\lambda}$ | $1/\lambda$ および $1/\lambda^2$ |
| 特殊アーラン†††<br>($n$ 個の指数分布の和) | $\frac{\lambda}{(n-1)!}(\lambda x)^{n-1}e^{-\lambda x},$<br>$x>0$ | $\frac{1}{(1+s/\lambda)^n}$ | $n\lambda$ および $n/\lambda^2$ |

†微分によってモーメントを計算するのにしばしば用いられる変換(**特性関数**(characteristic function)と呼ばれる)が $E(e^{iuX})=e^{ium-u^2\sigma^2/2}$ で与えられる．

††明らかに，指数分布は $n=1$ である特殊アーラン分布によって定義される．

†††**訳注**　$n$ 個の指数分布の和である特殊アーラン分布に対して，一般アーラン分布は直列および並列に接続された指数分布のネットワークとして定義される(第4章，演習問題4.16を参照せよ)．

は，自明の方法で2個よりも多くの独立な確率変数の和に拡張される．次の特殊な場合は特に重要である．すなわち，$n$ 個の独立な，相等しい分布に従う確率変数 $X_1, X_2, \cdots, X_n$ の和である．$A(z)$ が各 $X_i$ の共通の母関数であるならば，和 $S = X_1 + X_2 + \cdots + X_n$ の母関数は $A^n(z)$ となる．

## A.2　ラプラス変換と中心極限定理

離散的分布に対して母関数が果たしている役割は，対象となる分布関数が一般の場合にはラプラス変換(Laplace transform)によって担われることになる．今，確率変数 $X$ は非負かつ連続的であるとし，しかも，その分布関数は $F(x) = \Pr[X \leq x]$，密度関数は $f(x) = dF(x)/dx$ であるとしよう．この分布のラプラス変換を $F^*(s)$ と書き，これを $F^*(s) = \mathrm{E}(e^{-sX})$，すなわち，

$$F^*(s) = \int_0^\infty e^{-sx} f(x) dx \tag{A.2.1}$$

と定義する．もっと一般の分布(本書では扱われない)の場合には，$F^*(s)$は，スチルチェス積分，すなわち，

$$F^*(s) = \int_0^\infty e^{-sx} dF(x) \tag{A.2.2}$$

を用いて定義される．さて，各モーメントは，このラプラス変換から下記によって容易に決定することができる．すなわち，

$$\mathrm{E}(X) = -\lim_{s\to 0} \frac{d}{ds} \mathrm{E}(e^{-sX}) \tag{A.2.3}$$

$$\mathrm{E}(X^2) = \lim_{s\to 0} \frac{d^2}{ds^2} \mathrm{E}(e^{-sX}) \tag{A.2.4}$$

一般には，

$$\mathrm{E}(X^n) = \lim_{s\to 0} (-1)^n \frac{d^n}{ds^n} \mathrm{E}(e^{-sX}) \tag{A.2.5}$$

(表 A.1-1 は本書で使われている変換の実例を含んでいる．)

それぞれ密度関数 $f(x)$ と $g(x)$ を持ち，非負，連続的，かつ，互いに独立な二つの確率変数である $X$ と $Y$ の和は，畳込み

$$h(x) = \int_0^x g(x-y) f(y) dy \tag{A.2.6}$$

で与えられる密度関数を持ち，しかも，そのラプラス変換は次式で与えられる．

$$H^*(s) = G^*(s)F^*(s) \tag{A.2.7}$$

一般性のある表し方では，$\mathrm{E}(e^{-s(X+Y)}) = \mathrm{E}(e^{-sX})\mathrm{E}(e^{-sY})$ と書かれる．この式は期待値の定義および $X$ と $Y$ の独立性から得られる．したがって，密度の畳込みは，ラプラス変換の演算では変換の積となる．これは，丁度，確率塊関数の畳込みが，$z$-変換の演算では母関数の積になるのと同じである．母関数の場合と同じく，この結果も任意個の互いに独立な確率変数の和に拡張される．同様に，特殊な場合で重要なのは，それぞれが同一の分布関数 $F(x)$ を持つ $n$ 個の互いに独立な確率変数の和 $S = X_1 + X_2 \cdots + X_n$ である．待ち行列システムにおける待ち時間の研究では，このような例がしばしば見られる．これに対しては，次式が成立する†．

$$\mathrm{E}(e^{-sS}) = [F^*(s)]^n \tag{A.2.8}$$

確率変数の和に関係する極めて重要な結果は，著名な中心極限定理(central limit theorem)である．$X_1, X_2, \cdots, X_n$ は，互いに独立であって，それぞれが，有限な1次および2次のモーメントを持つ同一の分布に従う確率変数の系列であるとする．今，

$$S_n = X_1 + X_2 + \cdots + X_n$$

と置き，$S^0{}_n$ を次のように定義する．

$$S^0{}_n = [S_n - \mathrm{E}(S_n)]/[\mathrm{Var}(S_n)]^{1/2}$$

古典的な形の中心極限定理は，$n \to \infty$ とすると，$S^0{}_n$ の分布は零平均と単位分散を持つ正規分布に近付くということを主張している．式で表現すると，

$$\lim_{n\to\infty} \Pr[S^0{}_n \leq x] = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-y^2/2} dy$$

任意の(相等しい)分布に従う確率変数の和の分布の取扱いは往々にしてひどく不便である．それに反し，中心極限定理が提供しているいわゆる正規近似(normal approximation)は取扱いの単純化に大いに役立っている．この近似の例解は初等確率論の教科書に譲るが，応用例を第5章と第7章に求めることはできよう．しかし，ここでは，下記の諸項目が包含されるように上記の結果を著

† たとえば，表 A.1-1 における指数分布の変換とアーラン分布の変換の関係に注意しよう．

しく拡張し得ることを指摘しておかなければならない．

1．互いに異なる分布を持つ確率変数
2．独立でない確率変数のある種のクラス
3．1次モーメントが存在しないような分布

この外にも，与えられた $n$ に対して正規近似の誤差を記述する問題について少なからぬ研究が為されている．

## A.3　マルコフ連鎖

確率変数の系列 $X_0, X_1, \cdots, X_n, \cdots$ を考える．各確率変数は，その値として，集合 $N=\{0, 1, 2, \cdots\}$† に含まれる整数を取り得るものと仮定しよう．全ての $n$ および $X_i$ の全ての可能な値に対して

(A.3.1)
$$\Pr[X_n=j|X_0=i_0, X_1=i_1, \cdots, X_{n-1}=i_{n-1}] = \Pr[X_n=j|X_{n-1}=i_{n-1}]$$

すなわち，$X_n$ の分布が，$X_{n-1}$ によって完全に決定され，$(n-1)$番目の段階に先立つ $\{X_n\}$ の振舞には無関係であるならば系列 $\{X_i\}$ は**マルコフ連鎖**(Markov chain)であるという．

$\{X_i\}$ によって記述されるシステムは，$X_n=j$ であるならば，$n$ 番目の**段階**(step)において**状態** $j$ にあるといわれる．このことに対応して，集合 $N$ をマルコフ連鎖 $\{X_i\}$ の**状態空間**(state space)と呼ぶ．また，分布 $\Pr[X_0=j]$ $(j\in N)$を**初期分布** (initial distribution) といい，条件付確率 $\Pr[X_{m+n}=j|X_m=i]$ $(n\geq 1)$を $n$ **段遷移確率**($n$-step transition probability)と呼ぶ．さらに，$X_{n-1}=i$ かつ $X_n=j$ であるならば，システムは $n$ 番目の段階で遷移 $i\to j$ が行われたという．もしも初期分布と1段遷移確率を知ることができるならば，各 $X_n(n\geq 1)$の分布を一意に決定することが可能になる．

本書における応用では．連鎖の指数(chain index)$n$ を時助変数(time parameter)であると解釈することができる．たとえば，$X_n$ はセルフサービスのエレベータ内の人数を表しているとしよう．この場合，時助変数 $n$ は，エレベータ内人数が変化する時刻$(t_0, t_1, \cdots, t_n, \cdots)$を指示する指数となる．あるいはまた，$t_i$ はエレベータが次々と各階に到達する一連の時刻を表すとすること

† 実は，一般的に，$X_i$ が取り得る値を任意の可付番集合の元であると仮定してもよいのであるが，実際上は非負整数の集合を仮定する方が明らかに便利である．なお，一般の場合に対応して全結果の書き直しができることは自明である．

も可能である．なお，$t_n$ は，しばしば，マルコフ連鎖 $\{X_n\}$ の**遷移時点**(epoch)と呼ばれる．

マルコフ連鎖 $\{X_n\}$ は，遷移確率 $\Pr[X_{m+n}=j|X_m=i]$ が $m$(または冒頭の遷移時点 $t_m$)の取り方に無関係であるならば，**一様**(homogeneous)であるといわれる．本書では，一様なマルコフ連鎖しか扱わないから，上記の($n$ 段)遷移確率を $p_{ij}^{(n)}$ と書くことができる．1段遷移確率は常に $p_{ij}$ と書かれる．

ある $n$ に対して，$p_{ij}^{(n)}>0$ であるならば，状態 $j$ は状態 $i$ から**到達可能**(reachable)であるという．$\{X_n\}$ の全ての状態が他の全ての状態から到達可能であるならば，$\{X_n\}$ は**既約**(irreducible)であるという．本書で扱う全てのマルコフ連鎖は既約である．**可約**(reducible)なマルコフ連鎖の解析は，元の状態空間において，互いに連絡がある状態，すなわち，互いに他から到達可能な状態だけから成る(極大)部分集合を全て考え，これらによって形成されるマルコフ連鎖を解析することによって実行することができる．

システムが初めに状態 $j$ にあると仮定し，$f_j^{(n)}$ は，システムが $n$ 番目の段階で状態 $j$(の占位)に初めて立ち帰る確率を表すものとする．システムがいつかは状態 $j$ に立ち帰る確率は，明らかに次の通りである．

$$f_i=\sum_{n=1}^{\infty}f_j^{(n)} \tag{A.3.2}$$

もし，$f_j=1$ であるならば，状態 $j$ は**再帰的**(recurrent)(または**固執的**(persistent))であるといい，$f_j<1$ であるならば，状態 $j$ は**非再帰的**(non-recurrent)(または**過渡的**(transient))であるという．もし，状態 $j$ への再帰が段階 $t, 2t, 3t, \cdots$ でだけ生じる可能性があり，$t>1$ がこの性質を持つ最大の整数であるならば，状態 $j$ は周期 $t$ を伴って**周期的**(periodic)であるという†．$t$ で割切れない全ての $n$ に対して $p_{jj}^{(n)}=0$ であることは明らかである．

もし，$f_j=1$ であるならば，

$$u_j=\sum_{n=1}^{\infty}nf_j^{(n)} \tag{A.3.3}$$

は状態 $j$ の**平均再帰時間**(mean recurrence time)を与える．特に，$u_j<\infty$ であるならば，$j$ は**非零再帰的**††(recurrent nonnull)(または**正再帰的**(recurrent positive))状態であるといい，$u_j=\infty$ であるならば，**零再帰的**††(recurrent

†**訳注**　明確な定義は次の通りである．状態 $j$ に対して，$p_{jj}^{(n)}>0$ となる正の整数 $n>0$ が存在するとする．このような $n$ の最大公約数を $d(j)$ と書き，$j$ の周期と呼ぶ．$d(j)>1$ のとき状態 $j$ は周期的であるといい，$d(j)=1$ のとき $j$ は非周期的(aperiodic)であるという．なお，周期 $d(j)$ の任意の整数倍 $md(j)$ について $p_{jj}^{(md(j))}>0$ が成り立つとは限らないことを注意しておく．

††**訳注**　非零再帰的状態は強エルゴード的，零再帰的状態は弱エルゴード的ともいわれる．

null)状態であるという．既約マルコフ連鎖では，1)全ての状態が非再帰的または零再帰的であるか，2)全ての状態が非零再帰的であるかのどちらかである．また，既約なマルコフ連鎖では，1)全ての状態が同じ周期を持つ周期的状態であるか，2)全ての状態が非周期的であるかのどちらかである．

次式

(A.3.4) $$\pi_j = \sum_{i=0}^{\infty} \pi_i p_{ij} \quad (j = 0, 1, 2, \cdots)$$

を満足する確率分布 $\{\pi_j\}$ は，$\{X_n\}$ の**定常分布**(stationary distribution)と呼ばれる．有限(すなわち，有限な状態空間を持つ)マルコフ連鎖に対して，式(A.3.4)は，しばしば，行列記法を用いて表現される．有限マルコフ連鎖の状態空間を集合 $\{0, 1, \cdots, k\}$ とし，$\boldsymbol{\pi} = \{\pi_0, \pi_1, \cdots, \pi_k\}$ と置く．$\mathsf{P} = ((p_{ij}))$ が遷移確率行列を表すとすると，式(A.3.4)を次のように書き表すことができる．

(A.3.5) $$\boldsymbol{\pi} = \boldsymbol{\pi}\mathsf{P}$$

もし，初期分布が定常分布と同一であるならば，$\Pr[X_n = j]$ は全ての $n \geq 0$ に対して互いに相等しい．

さて，下記の基本的諸結果が知られている．

**定理 A.1**

既約な非周期的マルコフ連鎖においては，極限確率 $\lim_{n\to\infty} \Pr[X_n = j] = \pi_j$ $(j \geq 0)$が常に存在し，しかも，初期分布に無関係である．さらに，次の二つの可能性が存在する．

1. 状態は全て非再帰的状態であるかまたは零再帰的状態である．この場合には，全ての状態 $j$ に対して $\pi_j = 0$ であり，定常分布は存在しない．
2. 全ての状態は非零再帰的状態である．この場合には，全ての状態 $j$ に対して $\pi_j = 1/u_j > 0$ であり，$\{\pi_j\}$ は唯一の定常確率分布を成す．この分布は，

(A.3.6) $$\sum_{j=0}^{\infty} \pi_j = 1$$

を満足するような連立方程式(A.3.4)の解として一意に決定される．この既約，非周期的，かつ(全ての状態が)非零再帰的なマルコフ連鎖を**エルゴード的**(ergodic)であるという．

**定理 A.2**

有限，非周期的，かつ，既約なマルコフ連鎖の全ての状態は非零再帰的状態である.

**定理 A.3**

状態空間 $\{0, 1, 2, \cdots\}$ を持つ非周期的かつ既約なマルコフ連鎖 $\{X_n\}$ は，式(A.3.4)を満足し，かつ，全ての項が零であるのではないような絶対収束数列$\{\pi_k\}$ が存在するならば，そのときに限り，唯一の定常分布を持つ.

# 補遺B　再帰時間

時点の系列 $\tau_1, \tau_2, \cdots, \tau_n, \cdots$ は，間隔 $\tau_{n+1} - \tau_n (n \geq 1)$ が互いに独立に，しかも相等しい分布関数 $F(x)$ に従って分布し，また，それらとは独立に間隔 $(0, \tau_1)$ が分布関数 $F_0(x)$ に従って分布するような再帰(または再生)過程(recurrent (renewal) process)における遷移時点を表す非負の確率変数であるとしよう．待ち行列システムにおける到着，機械の故障の発生，等々は再帰的事象(recurrent event)であることもある．簡単のため，$F(x)$ は密度 $f(x)$ を持つと仮定し，その平均値を $\alpha$ で表すことにする．

ある与えられた時刻 $t \geq 0$ に対し，次の事象の発生までの残りの時間の分布関数 $R_t(x)$ を求めたいとする．このような時間間隔は，**前進再帰時間**(forward recurrence time)と呼ばれるが，信頼性に関連するときは，**余生時間**(residual lifetime)ともいわれる．さて，前進再帰時間は，ある $n \geq 1$ に対して $t < \tau_n \leq t + x < \tau_{n+1}$ であるとき，しかも，そのときに限り，$x$ 以下であるということになる．したがって，

$$R_t(x) = \sum_{n=1}^{\infty} \Pr[t < \tau_n \leq t + x < \tau_{n+1}] \tag{B.1}$$

ところで，$\tau_n = u$ であるとすると，$\tau_{n+1} > t + x$ である確率は $1 - F(t + x - u)$ であることになるから，式(B.1)を次のように書くことができる．

$$R_t(x) = \sum_{n=1}^{\infty} \int_t^{t+x} [1 - F(t + x - u)]\, dH_n(u) \tag{B.2}$$

ただし，$H_n(u)$ は $n$ 番目の事象発生の時刻の分布関数である．ここで，$H(u) = \sum_{n=1}^{\infty} H_n(u)$ と置こう．式(B.2)における和と積分を交換することにより次式が得られる．

(B.3) $$R_t(x) = \int_t^{t+x} [1 - F(t + x - u)]\, dH(u)$$

次に，関数 $H(u)$ を計算しよう．

$H_n(u) = \Pr[\tau_n \leq u]$ は，単に，間隔$(0, \tau_1)$, $(\tau_1, \tau_2)$, $\cdots$, $(\tau_{n-1}, \tau_n)$の和が $u$ より大ではない確率であるということは直ちに理解されるであろう．ここで，ラプラス変換，したがって，補遺 A の結果を用いると，

$$H_n^*(s) = F_0^*(s)\,[F^*(s)]^{n-1}$$

が求められる．したがって，

(B.4) $$H^*(s) = \frac{F_0^*(s)}{1 - F^*(s)}$$

ところで，$F_0$ を次式のように定義すると，大変に具合のよいことになる．

(B.5) $$F_0^*(s) = \frac{1 - F^*(s)}{\alpha s}$$

というのは，この場合には，次式が成り立つからである．

(B.6) $$H^*(s) = \frac{1}{\alpha s}$$

上記の $F_0^*(s)$ の選択は，実際，本問題自体の動機付けから観ても望ましいものであるということが後述によって納得されるであろう．式(B.6)から $dH(u) = du/\alpha$が得られるから，これを式(B.3)に代入すると，少々の変形の後に次式のように簡単化される．

(B.7) $$R_t(x) = \frac{1}{\alpha} \int_0^x [1 - F(y)]\, dy$$

これが求めようとしていた結果である．$R_t(x)$ の期待値は，(部分積分法によって)容易に計算することができ，(4.2.27)で与えられた式になる．

$F_0(x)$ がいかなるものであっても，$t \to \infty$ としたときの $R_t(x)$ の極限的振舞は依然として式(B.7)で与えられることが証明可能である†．一方，分布 $R_t(x)$ のラプラス変換は正に式(B.5)に一致することが常道の方法で示される．したがって，$F_0(x)$ を上記のように定めたことは次のことを意味する．すなわち，過程がこれまでに無限の時間にわたって作動してきているときに，つ

---

† Cox, D. R., 再生理論. Methuen, London, 1962

まり，過程が統計的平衡状態にあるときに無作為(ランダム)に選んだ時刻が時間の原点であると仮定したのと同じであることを意味する．しかし，もともと(第4章と第5章では) $R_t(x)$ の平衡状態における値を求めているのであるから，$F_0(x)$ をこのように定めたことは適切である．もちろん，この点においてのみここで計算された $R_t(x)$ が4.2節で導入された $R(x)$ に厳密に等しいということができるのは当然である．

与えられた $t>0$ に対し，**後退再帰時間**(backward recurrence time)は，最後の事象以後の経過時間であると定義される．後退再帰時間の分布は，同じ初期条件を仮定すると(あるいは，ここでも，平衡状態分布のみを考えると)，全く類似の解析によって式(B.7)に等しいことが示し得る．

# 索　　引

## あ



## い



## う



## え

## お



## か



## き

## く



## け



## こ



## さ

## し

## ち



## つ・て



## と



## な



## に



## の

## は



## ひ



## ふ



## へ

## ほ



## ま



## み



## む



## め・も・ゆ

## よ



## ら



## り



## れ



## ろ・わ

# 英 字 索 引

Z

M E M O

MEMO

MEMO

《訳者略歴》

**川口喜三男**（かわぐち きみお）

昭和 33 年　大阪大学理学部数学科卒業
昭和 36 年　大阪大学工学部通信工学科卒業
昭和 36 年　富士通信機製造株式会社（現富士通株式会社）入社
昭和 43 年　大阪大学基礎工学部講師
昭和 50 年　大阪大学大型計算機センター助教授
昭和 51 年　名古屋工業大学情報工学科（現電気情報工学科）教授
**現　　在　名古屋工業大学電気情報工学科教授・工学博士**

**藤井　護**（ふじい まもる）

昭和 37 年　大阪大学工学部電子工学科卒業
昭和 39 年　大阪大学大学院工学研究科修士課程修了
昭和 39 年　三菱電機株式会社入社
昭和 42 年　大阪大学基礎工学部制御工学科助手
昭和 46 年　大阪大学基礎工学部制御工学科講師，大阪大学大型計算機センター勤務
昭和 51 年　大阪大学大型計算機センター助教授
昭和 62 年　大阪大学基礎工学部情報工学科助教授
**現　　在　大阪大学基礎工学部情報工学科助教授・工学博士**


**オペレーティング・システムの理論**

昭和 62 年 4 月 15 日　初版印刷
**昭和 62 年 4 月 25 日　初版発行**



検印廃止

訳　者　川　口　喜　三　男
　　　　藤　井　　　　　護

発行者　村　越　勝　弘

発行所　**日本コンピュータ協会**
東京都文京区本郷 3 丁目 5-5
〒113　電話 (03) 813-6929 (代)

乱丁本・落丁本はお買い求めの書店にておとりかえいたします．　（中央印刷／松岳社）
ISBN 4-88917-030-8　C 3055
Printed in Japan